HACK X CRACK: HACKEANDO SERVIDORES WINDOWS EN 40 SEGUNDOS :)

PC PASO a PASO

CURSO DE REDES TCP-IP DESDE CERO

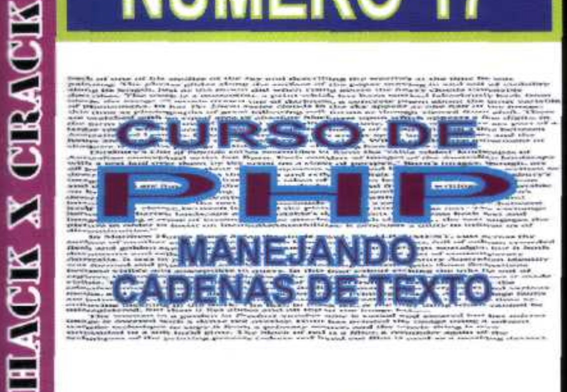


3 SERVIDORES ON LINE PARA TUS PRACTICAS DE HACK

NUMERO 17

LOS CUADERNOS DE HACK X CRACK

www.hackxcrack.com



TCP / IP: EL ALMA DE INTERNET



WINDOWS ADVANCED SERVER: HACKEADO EN 40 SEGUNDOS !!!



LO QUE NINGUNA REVISTA SE ATREVE A EXPLICAR

Nº 17 -- P.V.P. 4,5 EUROS



HACK X CRACK - HACK X CRACK - HACK X CRACK

LOS MEJORES ARTÍCULOS GRATIS EN NUESTRA WEB

PC PASO A PASO: TCP-IP NUNCA FUE TAN FACIL, ATREVETE!!!





EDITORIAL: EDITOTRANS S.L.
C.I.F: B43675701
PERE MARTELL Nº 20, 2º - 1ª
43001 TARRAGONA (ESPAÑA)

Director Editorial
I. SENTIS
E-mail contacto
director@editotrans.com
Título de la publicación
Los Cuadernos de HACK X CRACK.
Nombre Comercial de la publicación
PC PASO A PASO
Web: www.hackxcrack.com
Dirección: PERE MARTELL Nº 20, 2º - 1ª.
43001 TARRAGONA (ESPAÑA)

Director de la Publicación
J. Sentís

E-mail contacto
director@hackxcrack.com

Diseño gráfico:
J. M. Velasco

E-mail contacto:
grafico@hackxcrack.com

Redactores
AZIMUT, ROTEADO, FASTIC, MORDEA, FAUST
ENTROPIC, MEIDOR, HASHIMUIRA, BACKBON
ZORTEMIUS, AK22, DORKAN, KMORK, MAIL
TITINA, SIMPSIM... ... ...

Contacto redactores
redactores@hackxcrack.com

Colaboradores
Mas de 130 personas: de España, de Brasil, de Argentina, de Francia, de Alemania, de Japón y algún Estadounidense.

E-mail contacto
colaboradores@hackxcrack.com

Imprime
I.G. PRINTONE S.A. Tel 91 808 50 15

DISTRIBUCIÓN:
SGEL, Avda. Valdeparra 29 (Pol. Ind.)
28018 ALCOBENDAS (MADRID)
Tel 91 657 69 00 FAX 91 657 69 28
WEB: www.sgel.es

TELÉFONO DE ATENCIÓN AL CLIENTE: 977 22 45 8
Petición de Números atrasados y Suscripciones (Srta. Genoveva)
HORARIO DE ATENCIÓN: DE 9:30 A 13:30
(LUNES A VIERNES)

NUMERO 17 -- PRINTED IN SPAIN
PERIOCIDAD MENSUAL
Deposito legal: B.26805-2002
Código EAN: 8414090202756

EDITORIAL

EL TIEMPO Y LAS PROMESAS

Otro mes, otro número de PC PASO A PASO // Los Cuadernos de Hack x Crack.

Este número marca el inicio de una nueva época para esta revista. Los cambios se dejarán ver poco a poco y el objetivo es ir cumpliendo algunas de las promesas que hicimos en anteriores entregas.

Para empezar iniciamos este mes el Curso de TCP/IP, algo esperado por muchos lectores y que ha costado mucho empezar. Espero sinceramente que os guste, porque nunca se ha escrito sobre TCP / IP de la forma en que nosotros lo hemos hecho.

También se nos ha reclamado que hagamos textos sencillos de hacking para los novatos. Muy bien, pues este mes te "merendarás" cuantos servidores Windows quieras en 40 segundos :)

Esperando poder cumplir contigo, nos esforzaremos cada mes en ser mejores y mantenernos en este mercado. Parece que Hack x Crack abrió la "veda" de "revistas de hacking"... pero seguimos convencidos de que solo una da "algo mas".

Nosotros no nos conformamos con darte "carnaza", queremos que APRENDAS como funcionan las cosas. Sabemos que es una línea editorial peligrosa, nuestros lectores decidirán si merecemos seguir en el mercado los próximos años.

Gracias de nuevo a nuestros lectores, a nuestros sufridos colaboradores y a la peña del foro.

GRACIAS A TODOS !!!!

INDICE

INDICE DE ANUNCIANTES

Hackeando un Servidor Windows en 40 SEGUNDOS!!!

- Vamos a Iniciar una sesión en un PC con Windows NT, Windows 2000, Windows 2003 o Windows XP sin conocer el PASWWORD. Lo haremos con poderes de administrador de sistema. Pofremos incluso cambiar el PASWORD del Sistema :)

1.- Presentación.

Antes que nada, saludo a todos los lectores de esta fantástica revista. Ya está, tenía que hacerlo, hace tiempo que la sigo cada mes y escribir este artículo es para mí un verdadero honor :)

Para ir tomando contacto, decir que hace casi 12 años que me dedico a la informática y empecé, como muchos en esta profesión, arreglando equipos en una empresa de Hardware. Vale, vale... quizás no te interese mi vida, pero unos cuantos años después me contrataron en otra empresa que se dedicaba a "solucionar" problemas informáticos en multitud de Bancos de toda España... ¿ya te intereso mas? Ja, ja... solo decirte que estuve en esta empresa durante 3 años y todavía me da miedo ingresar dinero en una cuenta bancaria ;p (por motivos obvios no daré más detalles sobre mi vida ;p)

Quizás te preguntes a qué viene todo esto... bueno, pues muy sencillo. Los bancos utilizan unos programas llamados ERPs. No es necesario para este artículo saber nada sobre los ERPs aunque ya se habló de ellos en anteriores números de la revista, simplemente recordaros que son complejos programas que utilizan las GRANDES EMPRESAS y muchos gobiernos para mantenernos a todos bien "controlados".

Una vez estuve en un edificio de tres plantas que contenía los equipos donde se albergaban los datos de una gran multinacional en España, hablo de tres plantas repletas de inmensos servidores de datos (quien no lo ha visto, no puede ni imaginárselo... que sensación... ...). Esos servidores de datos son (valga la redundancia) los que sirven los datos a cajeros automáticos, a los señores de la ventanilla del banco y hoy en día los que gestionan también la banca on-line. Imaginad lo complejo que es el software encargado de todo eso (pues eso es un ERP).

2.- La historia.

Bueno, que me pierdo y no hemos ni tan siquiera empezado. Los ERPs conviven (cada vez más) con los sistemas que todos conocemos, si, hablamos de Windows y, en este caso, hablaremos concretamente de Windows NT, Windows 2000 y Windows XP.

En "los Windows" se instalan los CLIENTES ERP y la mayoría de directores de banco tienen un flamante equipo informático con Windows instalado y desde el que consultan los datos de "sus clientes".

Un buen día (sobre las 16:00 horas) me llamó el director de una pequeña sucursal y me rogó (se le notaba bastante desesperado) que me pasase por su oficina, que había cambiado la clave del Ordenador y que quizás la anotó mal porque era incapaz de iniciar el Windows.

Yo hacía ya tres meses que ya no trabajaba en la empresa encargada de esto y me extrañó que me llamase a mí. Le dije que llamase a la empresa y le enviarían a alguien. Me contestó que no, que le habían enviado a un "idiota" (palabras textuales) que quería reinstalarle el

sistema y que perdería sus datos personales (casi tres años de contactos personales y esas cosas que los directores de banco guardan en sus equipos, si, desde pornografía hasta recetas de cocina, he visto de todo).

Le comenté que el protocolo de trabajo exige que esto sea así, por motivos de seguridad. Pero el hombre estaba realmente desesperado, me repetía sin cesar que su agenda personal estaba allí y que tenía datos muy importantes que necesitaba para esa misma tarde.

Le dije que le explicase todo eso a la empresa, que había maneras de recuperar la información; pero me respondió que, según le había asegurado el Director Técnico, si había perdido la clave no había manera de Iniciar Sesión en el equipo, que lo máximo que podía hacer es sacar el disco duro y en tres semanas (mas o menos) le diría si había podido recuperar la información de su agenda.

Me repitió (una y otra vez) que necesitaba acceder no solo a su agenda, sino a un montón de archivos personales y que no quería que nadie "husmease" (palabras textuales) en sus archivos.

Vale, finalmente me convenció para que le echase un vistazo (nunca viene mal un director de banco "amigo"). Cogí mis herramientas de trabajo ("las mías", las que ahora os enseñaré) y en poco menos de una hora ya estaba frente al director y su flamante PC.

Iniciamos el equipo y apareció la típica ventanita de Inicio de Sesión donde te pide la clave de usuario



Probamos (durante 5 minutos) lo típico, poner la contraseña en mayúsculas y en minúsculas y esas cosas con resultado negativo. Le insinué que podía intentar hackearle la cuenta... consejo: insinua, pero nunca le digas a nadie que sabes hacerlo y que sabes perfectamente que puedes solucionar su problema, si después no funcionase te odiarán el resto de su vida por dar falsas esperanzas.

Me miró sorprendido, como si estuviese planteándole algo de ciencia ficción. Me estudió con una mirada inquieta y me preguntó ¿Cuántas horas necesitas? ... le contesté con una sonrisa mientras introducía un diskette en el PC y lo reiniciaba.

40 segundos después, el director estaba abriendo su valiosa agenda electrónica y mi ayuda se vio recompensada con 50.000 pesetas de las de antes ;) Nunca he visto a nadie tan.... ¿agradecido?... no... su cara era de verdadera INCREDULIDAD.

Eso es lo que aprenderemos a hacer hoy, entrar en cualquier Windows NT, 2000 o XP sin conocer la contraseña. De hecho, lo que haremos realmente es cambiar (o eliminar) la contraseña :)



He podido ver...

He podido ver muchas veces que en bancos y empresas tienen PCs fácilmente accesibles por casi todo el mundo, por ejemplo personal de limpieza (e incluso clientes en la sala de espera).

La gente cree que un PC que tiene una contraseña está protegido... ahora veremos que CUALQUIERA puede entrar en él: Queda todo el mundo avisado!!!

3.- PREPARANDO NUESTRAS ARMAS

Vamos a preparar nuestras "herramientas" para poder entrar en cualquier Windows

NT3.51, Windows NT4, Windows 2000, Windows XP y Windows Server 2003. Debo advertir yo no considero esto una técnica de hacking, de hecho, creo que todo administrador de sistemas debería conocer estas técnicas.

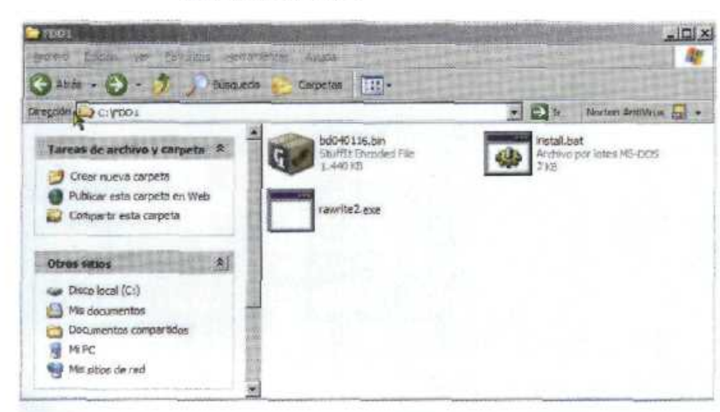
Primero iremos a la Web de esta revista (www.hackxcrack.com) , entraremos en la sección **Artículos Liberados y Descargas,** pulsaremos sobre la revista número 17 y nos descargaremos el archivo **bd040116.zip**



Por si acaso...

Por si acaso la Web de la revista no está ON LINE, al final de este artículo os pondré otros sitios para descargaros los archivos que utilizaremos ;)

Descomprimiremos el archivo **bd040116.zip** donde queramos (nosotros lo haremos en C:\FDD1) y obtendremos tres curiosos archivos: **bd040116.bin**, **install.bat** y **rawrite2.exe**

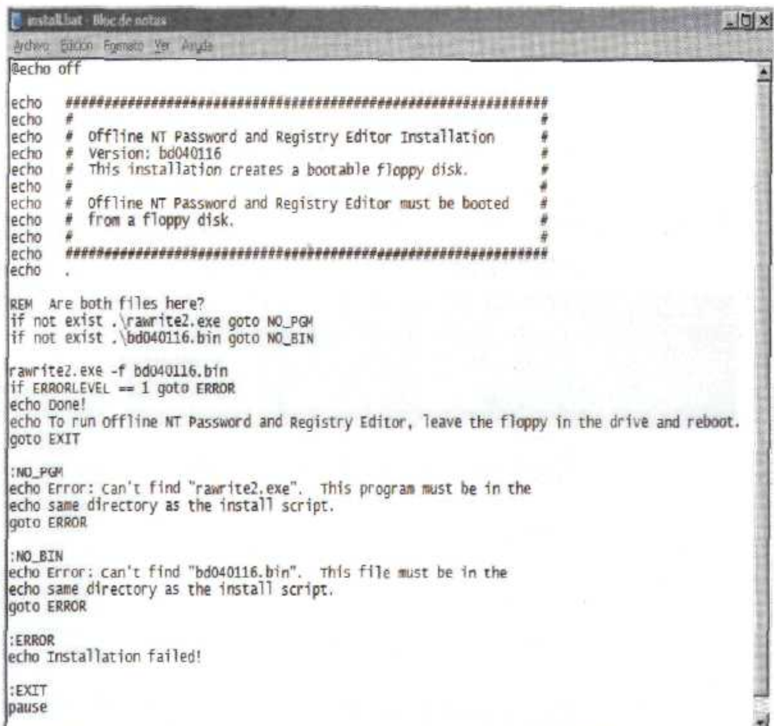


bd040116.bin: Esta es una imagen de diskette. Supongo que debes estar harto de grabar imágenes de CD con la extensión ISO o NRG, pues bien, por si no lo sabes, también existen las imágenes de Diskettes :) ... ... ahora veremos que la diskettera (unidad a:) *sirve para algo mas que estorbar :)*

rawrite2.exe: Este es el programa que utilizaremos para grabar la imagen del diskette. Para grabar una imagen de CD utilizamos, por ejemplo, el Nero Burning ROM, pues bien, para grabar una imagen de diskette utilizaremos el rawrite :)

install.bat: esto es un simple archivo ejecutable que automáticamente llamará al programa rawrite2.exe para que nos copie la imagen *bd040116.bin en un diskette.*

Para el que no lo sepa, el archivo install.bat (y cualquier bat) puede visualizarse y editarse, por ejemplo, con el bloc de notas de Windows. Venga, pulsamos el botón derecho del Mouse sobre el archivo install.bat --> abrir con --> Bloc de Notas. Lo que veremos es esto:



```
@echo off

echo   ####################################################################
echo   #                                                                  #
echo   #  Offline NT Password and Registry Editor Installation            #
echo   #  Version: bd040116                                               #
echo   #  This installation creates a bootable floppy disk.               #
echo   #                                                                  #
echo   #  Offline NT Password and Registry Editor must be booted          #
echo   #  from a floppy disk.                                             #
echo   #                                                                  #
echo   ####################################################################
echo   .

REM  Are both files here?
if not exist .\rawrite2.exe goto NO_PGM
if not exist .\bd040116.bin goto NO_BIN

rawrite2.exe -f bd040116.bin
if ERRORLEVEL == 1 goto ERROR
echo Done!
echo To run Offline NT Password and Registry Editor, leave the floppy in the drive and reboot.
goto EXIT

:NO_PGM
echo Error: Can't find "rawrite2.exe".  This program must be in the
echo same directory as the install script.
goto ERROR

:NO_BIN
echo Error: Can't find "bd040116.bin".  This file must be in the
echo same directory as the install script.
goto ERROR

:ERROR
echo Installation failed!

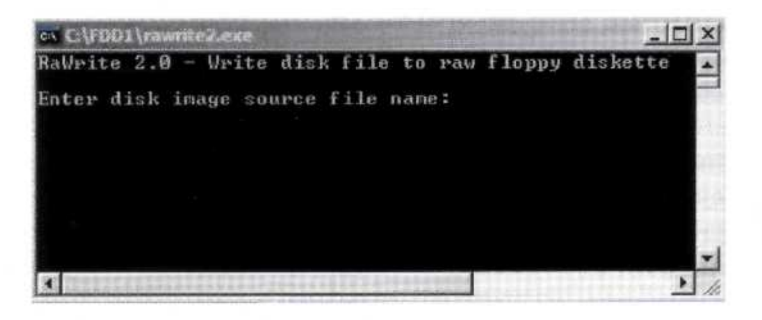
:EXIT
pause
```

Quien esté acostumbrado a la línea de comandos sabe perfectamente lo que hacen esas instrucciones, ahora no vamos a ponernos a dar un curso de MS-DOS. Como ya hemos dicho, simplemente nos creará un diskette.

Pues venga, vamos a grabar el diskette *"mágico"*.

1.- Ponemos un diskette en la diskettera (je, je... vale, vale, no me mates)

2.- Picamos dos veces sobre el archivo install.bat y nos saldrá esta pantallita



3.- nos pide que introduzcamos el nombre de la imagen de FDD a grabar, pues muy fácil, escribimos **bd040116.bin** y pulsamos enter.

4.- Ahora nos pregunta dónde vamos a grabarlo. No, no es que el programa sea tonto y no sepa donde está la diskettera... lo que pasa es que podríamos tener 2 o más disketteras y claro, nos pregunta en cual queremos grabarla. Pues venga, escribimos **a:** y pulsamos enter. Nos dirá que introduzcamos un diskette en la unidad a: (eso ya lo hicimos antes), por lo tanto volvemos a pulsar enter :)

```
C:\FDD1\rawrite2.exe
RaWrite 2.0 - Write disk file to raw floppy diskette

Enter disk image source file name: bd040116.bin
Enter target diskette drive: a:
Please insert a formatted diskette into drive A: and press -ENTER- :_
```

Empezará la grabación del diskette y cuando acabe se cerrará la "ventanita negra" :)

5.- Listo!!! Saca el diskette y déjalo a mano.

4.- UTILIZANDO NUESTRAS ARMAS:

PASO 1:

Ahora mismo tenemos un Diskette de arranque que iniciará una mini-versión de Linux. Pues venga, **reiniciamos el PC con el diskette dentro** y veremos que se inicia el Mini-Linux :)

Advertencia

No tengas miedo, no cambiará ningún archivo de tu PC, no cambiará ABSOLUTAMENTE NADA de tu PC hasta que YO TE LO DIGA.

Llegará un momento en que el Mini-Linux escribirá en tu disco duro y modificará tu sistema, PERO YO TE LO ADVERTIRÉ ANTES. De esta forma podrás cancelar la operación si así lo deseas.

Te recomiendo hacer esta practica completa en un pc antiguo que tengas por ahí perdido o en una máquina virtual (si te interesa eso de las máquinas virtuales quizás escriba un artículo sobre ello para el próximo número, estate atento a la revista :) Te digo esto porque, aunque no tiene por qué pasar nada malo, en la vida no puedes estar seguro de nada. Yo no me hago responsable de los destrozos que puedas causar en tu equipo ;p

equipo, al iniciarse leerá el diskette. Como el diskette es "de inicio" empezará a cargar su contenido, saldrán un montón de letras en la pantalla (en perfecto ingles, por supuesto) y finalmente nos encontraremos con un listado de nuestros discos duros así como sus particiones... y una invitación a que elijamos uno de ellos "SELECT:"

```
Initialization complete!

**************************************************************************
* Win/NT Registry Edit Utility Floppy / chntpw                           *
* (c) 1997 - 2004 Petter N Hagen - pnordahl@eunet.no                     *
* See file named "license" on floppy for licensing info and credits      *
*                                                                        *
* This utility will enable you to change or blank the password of        *
* any user (incl. administrator) on an Windows NT/2k/XP installation     *
* WITHOUT knowing the old password.                                      *
* Unlocking locked/disabled accounts also supported.                     *
*                                                                        *
* It also has a registry editor, and there is now support for            *
* adding and deleting keys and values.                                   *
*                                                                        *
* Tested on: NT3.51 & NT4: Workstation, Server, PDC.                     *
*            Win2k Prof & Server to SP4. Cannot change AD.               *
*            XP Home & Prof: up to SP1                                   *
*            Win 2003 Server (all?): Seems to work                       *
*                                                                        *
* HINT: If things scroll by too fast, press SHIFT-PGUP/PGDOWN ...        *
**************************************************************************

==========================================================
There are several steps to go through:
- Disk select with optional loading of disk drivers
- PATH select, where are the Windows systems files stored
- File-select, what parts of registry we need
- Then finally the password change or registry edit itself
- If changes were made, write them back to disk

DON'T PANIC! Usually the defaults are OK, just press enter
             all the way through the questions

==========================================================
* Step ONE: Select disk where the Windows installation is
==========================================================
Disks:
Disk /dev/ide/host0/bus0/target0/lun0/disc: 1610 MB, 1610612736 bytes
NT partitions found:
 1 : /dev/ide/host0/bus0/target0/lun0/part1  1533MB Boot

Please select partition by number or
a = show all partitions, d = load new disk drivers
l = relist NTFS/FAT partitions, q = quit
Select: [1] _
```

En nuestro caso, podemos ver que solo ha encontrado un disco duro de 1610 Megas:
Disks:
Disk /dev/ide/host0/bus0/target0/lun0/disc: 1610MB 1610612736 bytes

Y una partición de arranque (boot)
NT partitions found:
1 - /dev/ide/host0/bus0/target0/lun0 /part1 1533MB Boot



Este artículo...

Este artículo no trata sobre discos duros, pero debemos aclarar algunos conceptos para los novatos :)

Como todo el mundo sabe, el disco duro es donde se aloja el Sistema Operativo y cualquier archivo con el que trabajemos. Podríamos tener unos cuantos, aunque lo más normal es que te encuentres solo uno o dos.

Antes de poder utilizar un disco duro, hay que darle formato. Si tenemos (por ejemplo) un disco duro de 200GB, podríamos crear en él una sola partición de 200GB o dos particiones de 100GB e incluso 4 particiones de 50GB cada una. En este último caso, en el Windows veríamos que tenemos 4 discos duros (unidades c: d: e: y f:) pero REALMENTE tenemos un solo disco duro con 4 particiones.

De las particiones que tenemos, debe existir una que sea "de arranque" (BOOT). No hace falta profundizar mas en el tema, siendo bastante bestias diremos que la partición BOOT es donde está instalado la vase del Sistema Operativo.

En este caso nuestro preciado diskette nos dice que tenemos un solo Disco Duro de 1610 MB y una sola partición de 1533 MB que además es de arranque (BOOT). Pues perfecto, ya sabemos que el Sistema Operativo está en la única partición de arranque existente (la 1).

Posiblemente te preguntes por qué el disco duro es de 1610MB y la partición de 1533MB. No te preocupes, cuando formateamos siempre perdemos "capacidad" de disco (no es el momento ahora de explicar eso).

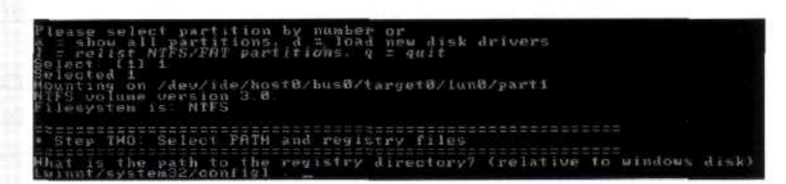
Lo que haremos con esta practica es cambiar (o borrar) la clave del Sistema Operativo Windows, es decir, modificaremos un archivo del Sistema Operativo Windows. Nuestra misión, por lo tanto, es encontrar la partición donde se aloja el Sistema Operativo Windows.

Deberemos encontrar una partición de arranque BOOT, puesto que tenemos un 98% de posibilidades de que el Sistema Operativo Windows esté en esa partición. En este caso es muy fácil, solo tenemos una partición y es de tipo BOOT. Pues ya está. Seguro que el Sistema Operativo está ahí, en la partición 1 ;p

PASO 2:

Pues venga, **pulsamos la tecla 1 en nuestro teclado y después la tecla enter**.

En este momento veremos que salen unos cuantos mensajes más. Simplemente se nos informa de que el Mini-Linux está "montando" la partición seleccionada (en este caso la partición 1) y de propina dos dice el sistema de archivos que contiene (en este caso NTFS versión 3).





Para el que...

Para el que esté muy verde, simplemente puntualizar que LINUX, *antes de poder leer y escribir sobre una partición* de un disco duro, debe "montarla" (hacerse con ella). Windows también lo hace, pero "en plan automático" (ni nos avisa ni nada).

Linux nos ha dicho que el sistema de archivos es NTFS. Si eres muy novato no tendrás ni idea de lo que es eso, pero bueno… cuando formateamos una partición podemos elegir entre varios sistemas de archivos, en Windows NT y Windows 2000 normalmente se selecciona el Sistema NTFS. Dejémoslo aquí ¿vale?, o podría empezar un curso sobre Sistemas de Archivo que acabaría durmiéndote :)

PASO 3:

Ahora, como podemos ver en la imagen

anterior, nos está pidiendo que introduzcamos la ruta para poder acceder al "registro" del Sistema Operativo Windows. Podemos ver que por defecto nos "insinúa" que podría estar en la ruta winnt/system32/config

Esa es la RUTA POR DEFECTO que elige Windows cuando se instala y que prácticamente nadie modifica. Tenemos un 99% de posibilidades de que sea correcta :)

Pues venga, **pulsamos enter** y nos aparecerá un listado de archivos.

```
========================================================
* Step TWO: Select PATH and registry files
========================================================
What is the path to the registry directory? (relative to windows disk)
[winnt/system32/config] :
-rw-------    1 0      0         24576 Feb 17 16:30 SAM
-rw-------    1 0      0         28672 Feb 17 18:40 SECURITY
-rw-------    1 0      0       2056192 Feb 17 18:32 SYSTEM.ALT
-rw-------    1 0      0        122880 Feb 17 18:28 default
-rw-------    1 0      0       6647808 Feb 17 20:20 software
-rw-------    1 0      0       2056192 Feb 17 18:32 system
-rw-------    1 0      0        143360 Feb 17 09:12 userdiff

Select which part of registry to load, use predefined choices
or list the files with space as delimiter
1 - Password reset [sam system security]
2 - RecoveryConsole parameters [software]
q - quit - return to previous
[1] : _
```

Paso 4:

Ahora nuestro diskette nos ofrece tres posibilidades. Vamos a dedicarnos a lo nuestro que es cambiar la clave del sistema, por lo tanto seleccionaremos la primera: Password Reset

Pues venga, **pulsamos 1 y después la tecla enter.**

```
Select which part of registry to load, use predefined choices
or list the files with space as delimiter
1 - Password reset [sam system security]
2 - RecoveryConsole parameters [software]
q - quit - return to previous
[1] : 1
Selected files: sam system security
Copying sam system security to /tmp

========================================================
* Step THREE: Password or registry edit
========================================================
chntpw version 0.99.2 040105, (c) Petter N Hagen
Hive's name (from header): <\SystemRoot\System32\Config\SAM>
ROOT KEY at offset: 0x001020

File size 24576 [6000] bytes, containing 5 pages (+ 1 headerpage)
Used for data: 206/17512 blocks/bytes, unused: 6/2808 blocks/bytes.
Hive's name (from header): <SYSTEM>
ROOT KEY at offset: 0x001020

File size 2056192 [1f6000] bytes, containing 498 pages (+ 1 headerpage)
Used for data: 38603/2032792 blocks/bytes, unused: 7/3368 blocks/bytes.
Hive's name (from header): <temRoot\System32\Config\SECURITY>
ROOT KEY at offset: 0x001020

File size 28672 [7000] bytes, containing 6 pages (+ 1 headerpage)
Used for data: 483/23336 blocks/bytes, unused: 4/1048 blocks/bytes.

* SAM policy limits:
Failed logins before lockout is: 0
Minimum password length        : 0
Password history count         : 0

<>========<> chntpw Main Interactive Menu <>========<>

Loaded hives: <sam> <system> <security>

  1 - Edit user data and passwords
  2 - Syskey status & change
  3 - RecoveryConsole settings

  9 - Registry editor, now with full write support!
  q - Quit (you will be asked if there is something to save)

What to do? [1] -> _
```

Paso 5:

Ahora nos encontramos con unas cuantas opciones más. Que diskette más interesante ¿no? Je, je... bueno, vamos a lo nuestro, las claves. Como podemos ver debemos seleccionar la opción 1: Edit user data and password.

Pues venga, **pulsamos 1 y después enter**

```
What to do? [1] -> 1

===== chntpw Edit User Info & Passwords ====

RID: 01f4, Username: <Administrador>
RID: 01f5, Username: <Invitado>, *disabled or locked*
RID: 03e9, Username: <IUSR_EQUIPO1>
RID: 03ea, Username: <IWAM_EQUIPO1>
RID: 03e8, Username: <TsInternetUser>

Select: ! - quit, . - list users, 0x<RID> - User with RID (hex)
or simply enter the username to change: [Administrador] _
```

Paso 6:

Ahora tenemos ante nosotros el listado de usuarios del Sistema Operativo Windows. Podemos ver que este Sistema tiene 5 Usuarios: Administrador, Invitado, IUSR-EQUIPO1, IWAM-EQUIPO1 y TsInternetUser..

Como nosotros somos muy ambiciosos, nos interesa cambiar (o eliminar) la clave del usuario más poderoso... si, si... como ya debes suponer, nos interesa el Administrador :)

Nuestro diskette mágico ya presupone que queremos ser administradores, je, je... así que tan solo deberemos pulsar enter (si quisiésemos cambiar -o eliminar- la clave de otro usuario deberíamos escribir el nombre de ese usuario y pulsar enter)

Pues venga, pulsamos enter ;p

```
Select: ! - quit, . - list users, 0x<RID> - User with RID (hex)
or simply enter the username to change: [Administrador]
RID     : 0500 [01f4]
Username: Administrador
fullname:
comment : Cuenta para la administración del equipo o dominio
homedir :

Account bits: 0x0210 =
[ ] Disabled        | [ ] Homedir req.    | [ ] Passwd not req. |
[ ] Temp. duplicate | [X] Normal account  | [ ] NMS account     |
[ ] Domain trust ac | [ ] Wks trust act.  | [ ] Srv trust act   |
[X] Pwd don't expir | [ ] Auto lockout    | [ ] (unknown 0x08)  |
[ ] (unknown 0x10)  | [ ] (unknown 0x20)  | [ ] (unknown 0x40)  |

Failed login count: 0, while max tries is: 0
Total  login count: 1

* = blank the password (This may work better than setting a new password!)
Enter nothing to leave it unchanged
Please enter new password:
```

Paso 7:

Ahora tenemos ante nosotros unos cuantos

datos respecto al usuario seleccionado, no hace falta explicarlo, no nos perdamos en el bosque (al menos no en este artículo), vamos a lo que nos interesa.

Abajo del todo se nos pide que introduzcamos un nuevo password. Podemos poner un nuevo password o simplemente poner un asterisco (*). Para asegurarnos pondremos un asterisco y de esta forma lo que hacemos es BORRAR el password del Administrador, es decir, cuando reiniciemos el Windows no necesitaremos introducir ninguna clave ;)

Pues venga, **ponemos un asterisco y pulsamos enter**

```
* = blank the password (This may work better than setting a new password!)
Enter nothing to leave it unchanged
Please enter new password: *
Blanking password!

Do you really wish to change it? (y/n) [n] _
```

Paso 8:

Ahora nos pregunta si realmente queremos cambiar el password, le diremos que si. Pues venga, **pulsamos la tecla "y" y después enter.**

```
Do you really wish to change it? (y/n) [n] y
Changed!

Select: ! - quit, . - list users, 0x<RID> - User with RID (hex)
or simply enter the username to change: [Administrador] _
```

- PERO... ¿no me dijiste que me avisarías antes de cambiar nada en mi ordenador? ERES UN CAP*U*LL*!!!

Tranquilo... que poco confías en mí... hay que ver... Todo lo que hemos hecho hasta ahora ha sido, por decirlo de alguna manera, en modo simulacro.

Ahora tenemos delante las últimas opciónes

Paso 10:

CUIDADO!!! LO QUE HAREMOS AHORA SI VA A MODIFICAR LOS ARCHIVOS DEL ORDENADOR DONDE ESTAS HACIENDO ESTA PRACTICA!!! TE DIJE QUE AVISARÍA ANTES ¿VERDAD? PUES YA ESTAS AVISADO!!!

Fíjate en la imagen anterior, parece que ya has cambiado el password PERO NO ES VERDAD. Es ahora cuando tienes la opción para hacerlo. Ahora tenemos varias opciones, pero la que nos interesa es volver al menú principal. Pues venga, **pulsamos la tecla de exclamación "!" (SHIFT + 1) y después enter.**

Nos aparecerá el menú principal de nuestro diskette, fíjate en la imagen siguiente:

```
Select: ! - quit, . - list users, 0x<RID> - User with RID (hex)
or simply enter the username to change: [Administrador] !

<>========<> chntpw Main Interactive Menu <>========<>

Loaded hives: <sam> <system> <security>

  1 - Edit user data and passwords
  2 - Syskey status & change
  3 - RecoveryConsole settings
      - - -
  9 - Registry editor, now with full write support!
  q - Quit (you will be asked if there is something to save)

What to do? [1] -> _
```

La opción que EJECUTARÁ LOS CAMBIOS EN NUESTRO ORDENADOR es la última, la opción **q** (quit). Pues venga, **pulsamos la tecla q y después enter**

Nos encontraremos ante la última pantallita:

```
What to do? [1] -> q

Hives that have changed:
 #  Name
 0  <sam> - OK

==========================================================
* Step FOUR: Writing back changes
==========================================================
About to write file(s) back! Do it? [n] : _
```

Paso 11:

Fíjate en la pantalla anterior, nos está pidiendo que confirmemos que realmente queremos hacer los cambios. ¿Estamos seguros? Pues claro que si.

Pues venga, **pulsamos la tecla "y" y después enter.** En este instante se modificarán los archivos de tu PC y veremos el proceso en la pantalla.

```
==========================================================
* Step FOUR: Writing back changes
==========================================================
About to write file(s) back! Do it? [n] : y
Writing sam

NOTE: A disk fixup will now be done.. it may take some time
Mounting volume... OK
Processing of $MFT and $MFTMirr completed successfully.
NTFS volume version is 3.0.
Setting required flags on partition... OK
Going to empty the journal ($LogFile)... OK
NTFS partition /dev/ide/host0/bus0/target0/lun0/part1 was processed successfully
NOTE: Windows will run a diskcheck (chkdsk) on next boot.
NOTE: this is to ensure disk integrity after the changes
***** EDIT COMPLETE *****

You can try again if it somehow failed, or you selected wrong
New run? [n] : _
```

Por muy poco ingles que sepas, puedes ver que todo ha funcionado por los OK que salen :)

Ya solo nos queda **sacar el diskette del PC y reiniciar el equipo.**

5.- MISION CUMPLIDA!!!

Nuestro equipo se iniciará normalmente y de nuevo aparecerá la ventanita para que introduzcamos nuestra clave. PERO en este caso no introduciremos ninguna contraseña, simplemente pulsaremos el botón ACEPTAR



Si todo ha ido bien, y estoy un 98% seguro de que si, Windows se iniciará sin quejarse ;P

6.- ¿40 segundos?... mentira... si eres rápido puedes hacerlo en 10 segundos!!!

Como puedes ver, a lo largo del artículo te hemos resaltado en negrita los pasos a seguir, vamos a hacer una carrera de 10 segundos. Una vez introducido el diskette y reiniciado el equipo:

(segundo 1) pulsamos la tecla 1 en nuestro teclado y después la tecla enter

(segundo 2) pulsamos enter

(segundo 3) pulsamos 1 y después la tecla enter

(segundo 4) pulsamos 1 y después la tecla enter

(segundo 5) pulsamos enter ;p

(segundo 6) ponemos un asterisco y pulsamos enter

(segundo 7) pulsamos la tecla "y" y después enter

(segundo 8) pulsamos la tecla de exclamación "!" (SHIFT + 1) y pulsaremos enter

(segundo 9) pulsamos la tecla q y después enter

(segundo 10) pulsamos la tecla "y" y después enter

7.- Mas cosas...

¿Y si el PC que quiero hackear no tiene diskettera? ¿Y si solo tiene CD-ROM?

Pues nada hombre, que tenemos soluciones para todo. Entonces te descargas el archivo **cd040116.zip** desde www.hackxcrack.com. Al descomprimirlo te saldrá una imagen ISO que podrás grabar con el NERO y ya tienes lo mismo pero en CD. Además, en este caso tendrás también los drivers SCSI.

¿Emmmm... drivers SCSI?

En este ejemplo, el PC víctima tenía el disco duro conectado mediante IDE, pero puedes encontrarte sistemas que tienen los Discos Duros conectados por SCSI. En ese caso el Mini-Linux no podrá "ver" esos discos duros porque necesitamos cargar el driver correspondiente. Si tienes la versión en CD todo está incluido :)



IDE...

IDE (el que te encontrarás en el 90% de los casos) y SCSI (muy caro y utilizado especialmente en servidores) son los dos sistemas de conexión de Disco Duro más extendidos, y hace poco ha salido un nuevo sistema llamado SATA (la evolución del IDE).

Ya se que explicar esto así es como mínimo inexacto, pero si quieres mas información busca en GOOGLE (www.google.com).

¿Y si el PC que quiero hackear solo tiene diskettera pero los discos duros están conectados por SCSI? ¡ENTONCES QUÉ!

Nada hombre, no te sulfures... je, je... entonces te descargas el archivo **sc040116.zip** de www.hackxcrack.com, lo descomprimes y metes su contenido en otro diskette. Podrás utilizas ese diskette para cargar los drivers SCSI.

8.- Zona de Descarga y ampliación de la información:

Si, por el motivo que sea, los archivos aquí mencionados no pudieses descargarlos de www.hackxcrack.com, pásate por http://home.eunet.no/~pnordahl/ntpasswd/

Además, encontrarás bastante más información (por si quieres ampliar conocimientos). Por supuesto en perfecto ingles ;p Un saludo a todos y hasta la próxima... y gracias por leerme ;)

EL GANADOR DEL SORTEO DE UN SUSE LINUX 9 DEL MES DE ENERO ES:

MILAGROS MATEO

PROGRAMACION EN GNU LINUX

PROGRAMACION DEL SISTEMA

EL SISTEMA I.P.C. (II)

el_chaman. Luis U. Rodriguez Paniagua

- Vamos a entrar de lleno en la esencia de Linux.
- Estudiaremos las Señanes. Semáforos. Tuberías. etc.
- ¿Cómo se comunican los precesos entre ellos? AVERIGUALO !!!

1. Presentación

Una vez visto qué son los procesos y cómo podemos crearlos desde nuestros programas, el paso lógico que debemos dar es el estudio de como podemos hacer que los procesos se comuniquen entre sí. Para poder tratar este tema de comunicación entre procesos tendremos que hacer hincapié en los siguientes temas:

Señales

El mecanismo de señales entre procesos es uno de los sistemas de comunicación entre procesos más elementales y básicos disponibles en los sistemas UNiX.

Semáforos

Aunque los semáforos no sean estrictamente un mecanismo de comunicación entre procesos, es muy importante que los veamos, dado que nos permitirán manejar procesos concurrentes en recursos en los que la concurrencia no es manejada directamente por el S.O. Para quien no sepa qué es la concurrencia, sirva por ahora mostrarla como el problema que surge en un entorno multiproceso cuando varios procesos desean acceder simultáneamente a un mismo recurso (disco, memoria,etc...)

Memoria Compartida

Será un mecanismo de comunicación entre procesos que permitirá que estos compartan una zona de memoria virtual de manera que puedan comunicarse entre sí mediante esta memoria común. El acceso a esta memoria común deberá de ser gestionado por el propio programador, por ejemplo mediante semáforos.

Colas de Mensajes

Las colas de mensajes serán unas estructuras de datos gestionadas por el núcleo en la que podrán leer y escribir varios procesos. A diferencia del mecanismo anterior donde el programador será el responsable de gestionar el acceso a la memoria compartida, en las colas de mensajes es el propio núcleo el encargado de resolver la concurrencia, por lo que su manejo será más sencillo, aunque exigen usar mensajes previamente formateados.

Tuberías

Uno de los primeros sistemas de comunicación disponible entre procesos UNiX y en otros sistemas operativos. Veros las tuberías con nombre (*named pipes*) y las tuberías sin nombre (*unamed pipes*).

Comunicaciones en red

Una vez que hayamos visto los mecanismos de comunicación entre procesos locales, comenzaremos a estudiar cómo podremos comunicar procesos que están situados en máquinas distintas.

2. Señales

2.1. ¿Qué son las señales?

Las señales son mecanismos de comunicación y de manipulación de procesos en UNiX, si bien el manejo de las mismas puede variar

ligeramente entre los distintos sabores UNiX (4.3BSD, System V, POSIX, ...). Para efectos prácticos nos centraremos aquí en las señales System V, que están disponibles en todos los UNIX. Si alguien quiere profundizar en los distintos tipos de señales, puede acudir a la bibliografía suministrada en español al final del artículo [MÁRQUEZ].

A pesar de que cualquier proceso puede mandar señales a cualquier otro proceso, lo normal es que sea el núcleo el que constantemente esté mandando determinadas señales a los distintos procesos indicando el estado del sistema y de la ejecución del proceso (eventos).

Este mecanismo de comunicación consistirá en el envío de un mensaje (llamado señal) a un determinado proceso con el fin de informarle de alguna situación especial.

Las señales son *asíncronas*. Esto quiere decir que cuando un proceso recibe una señal, el proceso la atiende *inmediatamente* dejando aparte cualquier otra tarea que estuviese realizando. La forma de atender esta señal será de tres maneras posibles:

Ignorándola, así, con dos bemoles. Esto no será siempre posible. Ya veremos que hay determinadas señales que no se pueden ignorar por las buenas.

Invocar a una rutina de tratamiento de señal por defecto. Esta rutina perteneciende al núcleo del S.O. realizará una determinada tarea en función del tipo de señal que usualmente suele ser la terminación del proceso de una manera más o menos barroca.

Invocar a una rutina de tratamiento de señal hecha por el programador.

2.2. Tipos de señal

Para tener una lista completa de las señales disponibles en nuestro sistema, y al evento que corresponde cada una de estas señales, podemos invocar a la página del manual correspondiente: man 7 signal (ojo, el 7 indica que es la sección 7 la que debemos mirar, no la que se muestra por defecto que es la correspondiente a la función *signal* del lenguaje C) obteniendo:

SIGNAL(7) Manual del Programador de Linux SIGNAL(7)

NOMBRE

signal - lista de las señales disponibles

DESCRIPCIÓN

Linux permite el uso de las señales dadas a continuación. Los números de varias de las señales dependen de la arquitectura del sistema. Primero, las señales descritas en POSIX.1.

Señal	Valor	Acción	Comentario
SIGHUP	1	A	Cuelgue detectado en la terminal de control o muerte del proceso de control
SIGINT	2	A	Interrupción procedente del teclado
SIGQUIT	3	C	Terminación procedente del teclado
SIGILL	4	C	Instrucción ilegal
SIGABRT	6	C	Señal de aborto procedente de abort(3)
SIGFPE	8	C	Excepción de coma flotante
SIGKILL	9	AEF	Señal de matar
SIGSEGV	11	C	Referencia inválida a memoria
SIGPIPE	13	A	Tubería rota: escritura sin lectores
SIGALRM	14	A	Señal de alarma de alarm(2)
SIGTERM	15	A	Señal de terminación
SIGUSR1	30,10,16	A	Señal definida por usuario 1
SIGUSR2	31,12,17	A	Señal definida por usuario 2
SIGCHLD	20,17,18	B	Proceso hijo terminado o parado
SIGCONT	19,18,25		Continuar si estaba parado
SIGSTOP	17,19,23	DEF	Parar proceso
SIGTSTP	18,20,24	D	Parada escrita en la tty
SIGTTIN	21,21,26	D	E. de la tty para un proc. de fondo
SIGTTOU	22,22,27	D	S. a la tty para un proc. de fondo

A continuación las señales que no están en POSIX.1 pero descritas en SUSv2.

Señal	Valor	Acción	Comentario
SIGBUS	10,7,10	C	Error de bus (acceso a memoria inválido)
SIGPOLL		A	Evento que se puede consultar (Sys V). Sinónimo de SIGIO

SIGPROF	27,27,29	A	Ha expirado el reloj de perfilado (profiling)
SIGSYS	12,-,12	C	Argumento de rutina inválido (SVID)
SIGTRAP	5	C	Trampa de traza/punto de ruptura
SIGURG	16,23,21	B	Condición urgente en conector (4.2 BSD)
SIGVTALRM	26,26,28	A	Alarma virtual (4.2 BSD)
SIGXCPU	24,24,30	C	Límite de tiempo de CPU excedido (4.2 BSD)
SIGXFSZ	25,25,31	C	Límite de tamaño de fichero excedido (4.2 BSD)

** Para los casos SIGSYS, SIGXCPU, SIGXFSZ y, en algunas arquitecturas, también SIGBUS, la acción por omisión en Linux hasta ahora (2.3.27) es A (terminar), mientras que SUSv2 prescribe C (terminar y volcado de memoria).

A continuación otras señales.

Señal	Valor	Acción	Comentario
SIGIOT	6	C	Trampa IOT. Un sinónimo de SIGABRT
SIGEMT	7,-,7		
SIGSTKFLT	-,16,-	A	Fallo de la pila en el coprocesador
SIGIO	23,29,22	A	E/S permitida ya (4.2 BSD)
SIGCLD	-,-,18		Un sinónimo de SIGCHLD
SIGPWR	29,30,19	A	Fallo de corriente eléctrica (System V)
SIGINFO	29,-,-		Un sinónimo para SIGPWR
SIGLOST	-,-,-	A	Bloqueo de fichero perdido.
SIGWINCH	28,28,20	B	Señal de reescalado de la ventana (4.3 BSD, Sun)
SIGUNUSED	-,31,-	A	Señal no usada.

** Aquí, - denota que una señal está ausente. Allí donde se indican tres valores, el primero es comúnmente válido para alpha y sparc, el segundo para i386, ppc y sh, y el último para mips. La señal 29 es SIGINFO /SIGPWR en un alpha pero SIGLOST en una sparc.

Las letras en la columna "Acción" tienen los siguientes significados:

A La acción por omisión es terminar el proceso.

B La acción por omisión es no hacer caso de la señal.

C La acción por omisión es terminar el proceso y hacer un volcado de memoria.

D La acción por omisión es parar el proceso.

E La señal no puede ser capturada.

F La señal no puede ser pasada por alto.

CONFORME A

POSIX.1

ERRORES

SIGIO y SIGLOST tienen el mismo valor. Este último está comentado en las fuentes del núcleo, pero el proceso de construcción de algunos programas aún piensa que la señal 29 es SIGLOST.

VÉASE TAMBIÉN

kill(1), kill(2), setitimer(2)

Linux 1.3.88 13 junio 1996 SIGNAL(7)

Como podemos ver, cada señal tiene asociado un valor determinado. Ese será el valor que manejen los procesos. En la siguiente tabla, sacada de [MÁRQUEZ]

Tabla 1. Señales UNIX System V

Nombre	Número	Generar core	Terminar	Ignorar	No se puede ignorar	Rutina por defecto
SIGUP	01		*			*
SIGINT	02		*			*
SIGQUIT	03	*	*			*
SIGILL	04	*	*			
SIGTRAP	05	*	*			
SIGIOT	06	*	*			*
SIGEMT	07	*	*			*
SIGFPE	08	*	*			*
SIGKILL	09		*		*	*
SIGBUS	10	*	*			*
SIGSEGV	11	*	*			*
SIGSYS	12	*	*			+
SIGPIPE	13		*			*
SIGALRM	14		*			*
SIGTERM	15		*			*
SIGUSR1	16		*			*
SIGUSR2	17		*			*
SIGCLD	18			*		*
SIGPWR	19			*		

2.3. Manejo de señales

Visto lo que son las señales y lo que podemos hacer con ellas, vamos a ver como utilizarlas. Nos ceñiremos en los ejemplos a la interfaz UNiX System V, prescindiendo de la

interfaz 4.3BSD dado que ésta es compatible con la anterior y además la mejora considerablemente. Para quien quiera profundizar en las llamadas 4.3BSD, puede obtener un listado de las páginas de manual referentes a las llamadas necesarias en http://www.ptf.com/dossier/TC/Proc_FrB_1.0.1.shtml. Estas páginas de manual están disponibles en cualquier *BSD

2.3.1. Envío de señales

Para enviar una señal de un proceso a otro se emplea la función kill disponible en el archivo de cabecera **signal.h**. El prototipo de esta función es el siguiente:

```
/////////////////////////////////////////////
// Página del manual: man 2 kill
/////////////////////////////////////////////
#include <sys/types.h>
#include <signal.h>
        int kill(pid_t pid, int sig);
```

Donde **pid** *identificará al proceso o conjunto* de procesos al que queremos enviar la señal. ¿Grupo de procesos? Hemos visto que el **PID** corresponde al número identificador de un proceso, ¿cómo es posible entonces que enviemos a un *grupo* de procesos una señal mediante el **PID**?

En nuestro caso el parámetro *pid* será un número entero que dependiendo del valor que reciba tendrá los siguientes significados:

Tabla 2. Valores de parámetro pid

pid > 0	pid corresponde al PID del proceso al que queremos enviar la señal.
pid = 0	La señal se envía a todos los procesos pertenecientes al mismo grupo del proceso que envía dicha señal.
pid = -1	La señal es enviada a todos los procesos cuyo identificador real es igual al identificador efectivo del proceso que lo envía. Si el proceso que lo envía tiene identificador de superusuario, la señal es enviada a todos los procesos del sistema salvo a los procesos 0 (swapper) y 1 (init).
pid < -1	La señal es enviada a todos los procesos cuyo identificador de grupo coincide con el valor absoluto (sin el signo -) de pid.

En el caso de que enviemos una señal a un proceso sobre el que nuestro proceso no tiene privilegios, la llamada **kill** falla.

El siguiente parámetro, sig, representa la señal que queremos enviar según la tabla y los valores arriba vistos. Si vale 0, se verifica que la función **kill** funciona sin enviar ninguna señal. Se suele utilizar para verificar la validez del valor de **pid**.

En caso de que la llamada a **kill** se realice correctamente, ésta retornará un 0; en caso de error, devolverá -1.

El siguiente ejemplo se muestra cómo utilizar la función **kill**

```
//////////////////////////////////////////////////////////////////////////////////////////
// ARCHIVO: kill.c
//
// DESCRIPCIÓN:
//   Este programa representa el funcionamiento del manejo
//   de señales mediante el uso de la función kill
//
//   Se compila como: gcc kill.c -o mi_kill
//////////////////////////////////////////////////////////////////////////////////////////

#include <stdio.h>
#include <signal.h>
#include <sys/types.h>
int main()
{
    int pid;
    // Creamos un nuevo proceso
    pid = fork();
    // Si somos el hijo...
    if (pid == 0)
    {
        // El proceso hijo imprime cada segundo un mensaje
        // hasta que le dejen...
        while(1)
        {
            printf("\n Me llamo %d y ", getpid());
            printf("tengo un yuyu que no veas.");
            sleep(1);
        }
    }
    else
```

```
    {
        // Somos el desalmado padre
        // Matamos a nuestro hijo enviándole la señal
        // SIGTERM
        sleep(5);
        printf("\n Yo soy %d y tengo un mal día.", getpid());
        printf("\n Asesinando un poco a %d.\n", pid);
        kill(pid, SIGTERM);
    }
    return 0;
}
```

Tras compilar y ejecutar este programa obtenemos una salida tal que:

```
luis@leonov:~/hxc/articulo10_el_chaman/codigo$ gcc kill.c -o mi_kill
luis@leonov:~/hxc/articulo10_el_chaman/codigo$ ./mi_kill

 Me llamo 12358 y tengo un yuyu que no veas.
 Me llamo 12358 y tengo un yuyu que no veas.
 Me llamo 12358 y tengo un yuyu que no veas.
 Me llamo 12358 y tengo un yuyu que no veas.

 Yo soy 12357 y tengo un mal dia.
luis@leonov:~/hxc/articulo10_el_chaman/codigo$
```

Lo que hemos hecho es sencillo pero ilustrativo: Creamos un proceso hijo que si le dejamos estará indefinidamente imprimiendo el mensaje ***Me llamo 12358 y tengo un yuyu que no veas*** cada segundo. La misión del padre será esperar cinco segundos (poca paciencia para ser un buen padre, ¿verdad?) y matar al proceso hijo mandándole la señal SIGTERM (Señal de Terminación).

Podemos observar que cuando enviamos una señal al proceso hijo este termina su ejecución. Pero también podemos hacer que ocurra otra cosa.

2.3.2. Tratamiento de señales

Imaginemos que en la situación anterior, al ser asesinado el hijo, éste quiera realizar alguna tarea previa, como informar al usuario, guardar datos de manera segura, escribir un testamento repudiando a su padre, etc.

El mecanismo del que vamos a disponer para realizar esta tarea es la función signal. Antes de meternos a explicar dicha función que es un poco enrevesada por su sintaxis, vamos a explicar la idea de su funcionamiento.

Esta función lo que hará es decirle a una señal que en vez de realizar el tratamiento que realiza por defecto cuando se le llama (el sistema es el encargado de ello), lo que hará es ejecutar una función que nosotros hayamos escrito previamente y en la que realizaremos las tareas que consideremos apropiadas.

Precisamente la tarea de signal será la de "enlazar" una determinada señal con una determinada función de tratamiento.

La sintaxis de signal según la página del manual es:

```
#include <signal.h>
void (*signal(int signum, void (*manejador)(int)))(int);
```

Tan espeluznante como aparenta. Esta sintaxis heredada de la familia BSD, puede ser escrita, según la página del manual como:

```
typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t manejador);
```

Claaaarooo. Ahorá está clariíiísimo. Un cristal, vamos...

Mantengamos la calma y utilicemos el sentido común. Sabemos como funciona typedef, por lo que podemos llegar a la conclusión de que typedef void (*sighandler_t)(int); nos está definiendo un tipo.

Puede parecer que como lo que va antes de sighandler_t es un void, representa al void.

Pero es que además va un * por lo que realmente representa a void *.

Ya hablamos de que void normalmente significa "*nada*" excepto cuando es un puntero, que se interpreta como *"un puntero que puede apuntarlo todo"*.

Luego podríamos pensar que *sighandler_t es un tipo que creamos nosotros y que representa a "*un puntero capaz de apuntar a cualquier zona de memoria*"...

Pero aún nos quedaría dar la última vuelta de tuerca: No hemos llamado a nuestro tipo *sighandler_t sino *sighandler_t(int). ¿Y qué representa eso? ¡Una función! Luego lo que estamos declarando en este tipo es *un puntero capaz de apuntar a cualquier función void y con un parámetro entero como primer parámetro*.

Visto esto ahora es *relativamente* sencillo entender el prototipo de sighandler_t signal(int signum, sighandler_t manejador); tenemos pues que signal es una función que recibe como parámetros signum que representa a la señal

a asociar, manejador que será un puntero a una función void (el nombre de la función es un puntero a dicha función) y nos devuelve otro puntero a otra función (la función por defecto encargada de realizar la acción para esa señal) o SIG_ERR en caso de error.

Para que nos termine de desaparecer el pánico antes de volver a las interioridades de esta función, vamos a ver un ejemplo de uso y terminar de convencernos de esta manera de que es algo sencillo.

Vamos a modificar el anterior programa de manera que cuando el padre mate al hijo, éste al menos proteste un poco. El código resultante es:

```
////////////////////////////////////////////////////////////////////////////////
// ARCHIVO: kill.c
//
// DESCRIPCIÓN:
//   Este programa representa el funcionamiento del manejo
//   de señales mediante el uso de la función kill y el uso
//   de funciones de tratamiento de señales
//
//   Se compila como: gcc kill2.c -o mi_kill2

#include <stdio.h>
#include <signal.h>
#include <sys/types.h>
// Esta es la función que sustituye
// al tratamiento por defecto de la señal
void tratamiento(int sennal)
{
    // Tratamos la señal a nuestro modo
    printf("\n Soy el proceso %d", getpid());
    printf("\n Se recibió la señal %d", sennal);
    printf("\n Ahora yo soy mi propio ángel exterminador\n");
    // DEBEMOS terminar el proceso hijo
    exit (0);
}
int main()
{
    int pid;
    // Asociamos la señal que empleábamos con el nuevo tratamiento
    if(signal (SIGTERM, tratamiento)==SIG_ERR)
    {
        printf("\n Error al asignar función de tratamiento\n");
    }
    // Creamos un nuevo proceso
    pid = fork();
    // Si somos el hijo...
    if (pid == 0)
    {
        // El proceso hijo imprime cada segundo un mensaje
        // hasta que le dejen...
        while(1)
        {
            printf("\n Me llamo %d y ", getpid());
            printf("tengo un yuyu que no veas.");
            sleep(1);
        }
    }
    else
    {
        // Somos el desalmado padre
        // Matamos a nuestro hijo enviándole la señal
        // SIGTERM
        sleep(5);
        printf("\n Yo soy %d y tengo un mal día.", getpid());
        printf("\n Asesinando un poco a %d.\n\n", pid);
        kill(pid, SIGTERM);
    }
    return 0;
}
```

Como vemos en la llamada a la función, signal (SIGTERM, tratamiento), el primer parámetro corresponde a la señal que va a recibir el proceso hijo y el segundo tan solo al nombre de la función que realiza la nueva tarea de tratamiento de la señal.

La salida será:

```
luis@leonov:~/hxc/articulo10_el_chaman/codigo$ gcc kill2.c -o mi_kill2
luis@leonov:~/hxc/articulo10_el_chaman/codigo$ ./mi_kill2

Me llamo 19208 y tengo un yuyu que no veas.
Me llamo 19208 y tengo un yuyu que no veas.
Me llamo 19208 y tengo un yuyu que no veas.
Me llamo 19208 y tengo un yuyu que no veas.
Yo soy 19207 y tengo un mal día.
Asesinando un poco a 19208.
Me llamo 19208 y tengo un yuyu que no veas.
Soy el proceso 19208
Se recibió la señal 15
Ahora yo soy mi propio ángel exterminador
```

Llamo la atención sobre un punto: Ahora el proceso hijo *se mata* porque así lo hemos requerido en la rutina de tratamiento de la señal (exit (0);). Pero, ¿qué ocurre si "olvidamos" escribir esa línea? ¿por qué? ¿Se podría crear de esta manera procesos "inmunes" a que los matasen? ¿Para qué? ¿Realmente serían "inmortales"?.

También es obligatorio citar que en ocasiones podremos hacer que la rutina de tratamiento de señales puede hacer que volvamos a estados por los que el proceso ya ha pasado.

2.3.3. Esperando una señal

En algunas situaciones nos interesará que un proceso se quede a la espera de recibir una señal. Esto puede suceder si, por ejemplo, el proceso necesita que algún trabajo sea hecho antes de proseguir.

Para ellos está la función pause() que detiene la ejecución de un determinado proceso hasta que este recibe una determinada señal.

A diferencia de otras funciones, pause() devuelve -1 si todo ha ido correctamente.

Como ejemplo vamos a ver el siguiente código en el que el, proceso hijo, espera a que el proceso padre le de permiso para hablar. Se observará que se han sustituido las llamadas a printf(... por llamadas a fprintf(stderr,.... La razón de ellos es que la salida de error es una salida *non buffered*; es decir, no es necesario que el buffer de salida de stderr esté lleno para imprimir su contenido. No así ocurre con stdout que es lo que usa printf haciendo que las líneas de salida se superpongan unas a otras.

```
//////////////////////////////////////////////////////////////////////////////////////////
// ARCHIVO: pause.c
//
// DESCRIPCIÓN:
//  Este programa representa el funcionamiento del manejo
//  de señales mediante el uso de las funciones kill y pause
//  así como el uso de funciones de tratamiento de señales
//
//  Se compila como: gcc pause.c -o mi_pause
//
// AUTOR:
//  Luis U. Rodríguez Paniagua
//  Este código se distribuye bajo los términos de la
//  Licencia Pública General (GPL) de GNU. Puede usted
//  consultarla en www.gnu.org
//////////////////////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <signal.h>
#include <sys/types.h>

// Esta es la función que sustituye
// al tratamiento por defecto de la señal SIGTERM
void tratamiento_1(int sennal)
{
      // Tratamos la señal a nuestro modo
      fprintf(stderr,"\n [%7d] Termina el proceso \n", getpid());
      // DEBEMOS terminar el proceso hijo
      exit (0);
}

// Esta es la función que sustituye
// al tratamiento por defecto de la señal SIGUSR1
// Presentará una frase
void tratamiento_2(int sennal)
{
      signal(sennal, SIG_IGN);
      fprintf(stderr,"\n\t [%7d] ¡Me han dejado hablar! ", getpid());
      signal(sennal, tratamiento_2);
}
int main()
{
      int pid, i;
      // Asociamos las señales que empleábamos con sus respectivos
      // tratamientos
      if ((signal (SIGTERM, tratamiento_1)==SIG_ERR) ||
         (signal (SIGUSR1, tratamiento_2)==SIG_ERR))
      {
            printf("\n Error en signal \n");
      }
      // Creamos un nuevo proceso
      pid = fork();
      // Si somos el hijo...
      if (pid == 0)
      {
            // El proceso hijo imprime cada segundo un mensaje
            // hasta que le dejen...
            while(1)
            {
                  fprintf(stderr,"\n [%7d] Yo calladito.", getpid());
                  pause();
            }
      }
      else
```

```
        {
            // Esperamos 4 segs a que los hijos se creen
             sleep(4);
            // Damos cinco órdenes de hablar
            for(i=0;i<5;i++)
            {
                fprintf(stderr,"\n [%7d] Mandando hablar....\n", getpid());
                kill(pid,SIGUSR1);
                sleep(3);
            }
            // matamos el proceso hijo
            kill(pid, SIGTERM);
        }
        return 0;
}
```

2.3.4. Recuperando estados anteriores del proceso

Arriba comentamos que era posible, mediante el empleo de funciones de tratamiento de señales, el hacer que un proceso volviese a alguno de sus estados anteriores. Esto puede ser aplicable también al resto de las funciones. Para poder hacer esto utilizaremos las funciones:

```
#include <setjmp.h>
int setjmp(jmp_buf env);
void longjmp(jmp_buf env, int val);
```

La misión de setjmp es la de guardar el estado del sistema en env. La primera vez que llamamos a setjmp, esta función retorna 0, pero el resto de las veces devolverá un valor que le será comunicado mediante longjmp.

La misión de longjmp es recuperar el estado almacenado en env, haciendo así que volvamos a la situación en la que nos encontrábamos cuando almacenamos el estado. El segundo parámetro de esta función, val, corresponde al valor distinto de 0 que retornará setjmp cuando recuperemos el estado.

En el siguiente ejemplo, se muestra como podemos hacer que un supuesto bucle que deba contar hasta cien, se reinicia constantemente a 0 tras contar hasta 4 o 5.

Debido a un mal funcionamiento en GNU/LiNUX de la función signal, se tiene que compilar este código de una manera un tanto especial, dado que debemos de pasar al compilador la macro LINUX para que usemos la función sysv_signal en vez de signal. En el resto de los sistemas operativos UNiX-like de los que dispongo (FreeBSD y Solaris) el comportamiento es el esperado. Más abajo pondré un ejemplo de buen y mal funcionamiento.

```
///////////////////////////////////////////////////////////////////////////////
// ARCHIVO: salto.c
//
// DESCRIPCIÓN:
//   Este programa muestra como recuperar un estado anterior
//   mediante el uso de las funciones setjmp y longjmp
//
//   Se compila como:
//       LiNUX: gcc -DLINUX salto.c -o mi_salto
//       otros: gcc  salto.c -o mi_salto
//
// AUTOR:
//   Luis U. Rodriguez Paniagua
//   Este código se distribuye bajo los términos de la
//   Licencia Pública General (GPL) de GNU. Puede usted
//   consultarla en www.gnu.org
///////////////////////////////////////////////////////////////////////////////

#include <stdio.h>
#include <signal.h>
#include <sys/types.h>
#include <setjmp.h>
#include <unistd.h>
// Variable global que representa el estado del sistema
// en un determinado momento.
jmp_buf env;
// Esta es la función que sustituye
// al tratamiento por defecto de la señal SIGTERM
void tratamiento_1(int sennal)
{
        // Tratamos la señal a nuestro modo
        fprintf(stderr,"\n [%7d] Termina el proceso \n", getpid());
        // DEBEMOS terminar el proceso hijo
        exit (0);
}
// Esta es la función que sustituye
// al tratamiento por defecto de la señal SIGUSR1
// En este caso se encargará de volver a un estado anterior.
void tratamiento_2(int sennal)
{
        fprintf(stderr,"\n Se produjo excepción; retornamos a estado\n");
        // Reasignamos esta función al tratamiento_2
```

```
    // para garantizar el correcto funcionamiento.
#ifdef LINUX
    if(sysv_signal(sennal, tratamiento_2)==SIG_ERR)
#else
    if(signal(sennal, tratamiento_2)==SIG_ERR)
#endif
    {
        printf("error al asignar señel en tratamiento 2");
        exit(-1);
    }
    else
    {
        longjmp(env, 10);
    }
}
int main()
{
    int pid, i,k=0,j;

    // Asociamos las señales que empleábamos con sus respectivos
    // tratamientos
#ifdef LINUX
    if ((sysv_signal (SIGTERM, tratamiento_1)==SIG_ERR) ||
      (sysv_signal (SIGUSR1, tratamiento_2)==SIG_ERR))
#else
    if ((signal (SIGTERM, tratamiento_1)==SIG_ERR) ||
      (signal (SIGUSR1, tratamiento_2)==SIG_ERR))
#endif
    {
        printf("\n Error en signal \n");
    }
    // Creamos un nuevo proceso
    pid = fork();
    // Si somos el hijo...
    if (pid == 0)
    {
        // El proceso hijo comienza a contar desde 0; establecemos un punto
        // de retorno de estado cuando k=0; asi nos aseguramos que cada vez
        // que llamemos a SIGUSR1, el proceso hijo reiniciará la cuenta.
        // Intentamos imprimir 100 números
        if (setjmp(env)==0)
        {
            fprintf(stderr,"\n[%5d] Estableciendo punto de retorno",getpid() );
        }
        else
        {
            fprintf(stderr,"\n[%5d] Volviendo al punto de retorno",getpid());
            k=0;
        }
        // Este bucle, EN TEORÍA, debe de contar hasta 100, pero no lo hará debido
        // a que recibirá del proceso padre distintas señales.
        for (j=0;j<100;j++)
        {
            fprintf(stderr,"\n [%5d] Contando: %3d", getpid(), k);
            k++;
            sleep(1);
        }
    }
    else
    {
        // Esperamos 4 segs a que los hijos se creen
        sleep(4);
        //  Enviamos 10 señales SIGUSR1 al proceso hijo
        for(i=0;i<10;i++)
        {
            fprintf(stderr,"\n [%5d] Mandando señal SIGUSR1 a [%5d]....\n", getpid(), pid);
            if(kill(pid,SIGUSR1)==-1)
                printf("error al enviar señal");
            sleep(5);
        }
        // matamos el proceso hijo
        kill(pid, SIGTERM);
    }
    return 0;
}
```

En un entorno GNU/LiNUX, la ejecución correcta será:

```
luis@leonov:~/hxc/articulo10_el_chaman/codigo$ gcc -DLINUX salto.c -o mi_salto
salto.c: En la función `tratamiento_2':
salto.c:48: aviso: comparación entre puntero y entero
salto.c: En la función `main':
salto.c:70: aviso: comparación entre puntero y entero
salto.c:71: aviso: comparación entre puntero y entero
luis@leonov:~/hxc/articulo10_el_chaman/codigo$  ./mi_salto

[24635] Estableciendo punto de retorno
 [24635] Contando:   0
 [24635] Contando:   1
```



```
[24635] Contando:  2
[24635] Contando:  3
[24634] Mandando señal SIGUSR1 a [24635]....

Se produjo excepción; retornamos a estado

[24635] Volviendo al punto de retorno
[24635] Contando:  0
[24635] Contando:  1
[24635] Contando:  2
[24635] Contando:  3
[24635] Contando:  4
[24634] Mandando señal SIGUSR1 a [24635]....
.................
[24635] Volviendo al punto de retorno
[24635] Contando:  0
[24635] Contando:  1
[24635] Contando:  2
[24635] Contando:  3
[24635] Contando:  4
[24635] Termina el proceso
```

Como vemos tras la compilación se obtienen unos mensajes de *warning* (advertencia) que podremos ignorar. El funcionamiento es el esperado. Sin embargo si en GNU/LiNUX tecleamos:

```
luis@leonov:~/hxc/articulo10_el_chaman/codigo$ gcc salto.c -o mi_salto
luis@leonov:~/hxc/articulo10_el_chaman/codigo$ ./mi_salto

[24682] Estableciendo punto de retorno
[24682] Contando:  0
[24682] Contando:  1
[24682] Contando:  2
[24682] Contando:  3
[24681] Mandando señal SIGUSR1 a [24682]....

Se produjo excepción; retornamos a estado

[24682] Volviendo al punto de retorno
[24682] Contando:  0
[24682] Contando:  1
[24682] Contando:  2
[24682] Contando:  3
[24682] Contando:  4
[24681] Mandando señal SIGUSR1 a [24682]....

[24682] Contando:  5
[24682] Contando:  6
[24682] Contando:  7
[24682] Contando:  8
[24682] Contando:  9
[24681] Mandando señal SIGUSR1 a [24682]....

................

[24682] Contando:  40
[24682] Contando:  41
[24682] Contando:  42
[24682] Contando:  43
[24682] Contando:  44
[24681] Mandando señal SIGUSR1 a [24682]....

[24682] Contando:  45
[24682] Contando:  46
[24682] Contando:  47
[24682] Contando:  48
[24682] Contando:  49

[ 24682] Termina el proceso
```

Que no es ni mucho menos el resultado esperado. Podemos de esta experiencia deducir que cuando sospechemos que nuestro programa no envía adecuadamente las señales mediante signal, tendremos que emplear sysv_signal.

Y con esto quedan vistas las señales. Como vemos los procesos no van a ser islas independientes, sino que se pueden comunicar entre ellos y realizar conjuntamente tareas más complejas. Si bien el empleo de señales es relativamente sencillo y muy útil, realizar determinadas tareas con ellas será a veces muy difícil o imposible. Por ello, más adelante veremos mecanismos más sofisticados que nos permitirán comunicar procesos de una forma mucho más potente.

3. Semáforos

3.1. Introducción

Si bien los semáforos no son estrictamente hablando mecanismos de comunicación, si entendemos ésta como el intercambio de datos entre procesos, su mayor utilidad sale a relucir precisamente en situaciones en las que varios

procesos intercambian información utilizando recursos comunes. Más adelante veremos como una situación clásica es la utilización de *Memoria Compartida* para intercambiar datos entre procesos. Entonces necesitaremos un mecanismo que permita o condone a cada proceso en un determinado momento el acceso a dicha memoria compartida para evitar serios problemas.

Fácil es deducir entonces, que los semáforos son mecanismos de *sincronización de procesos* en vez de mecanismos de comunicación (tal y como lo hemos visto). Pero tampoco podemos ignorar la visión del sistema de semáforos como una manera de pasarse señales entre procesos, como si fueran caballeros muy educados, que solicitan y ceden permiso al acceso a determinados recursos.

3.2. Funcionamiento de los semáforos

Ya por el nombre nos podemos hacer una idea de cómo funcionará un semáforo: En determinado momento, cuando queramos acceder a un recurso compartido (cruce de una calle), deberemos de consultar el estado del semáforo para saber si podemos acceder a él o no (cruzar o no). Tal y como sucede en los semáforos reales, si ignoramos lo que el semáforo nos indica, podemos provocar una catástrofe.

Más formalmente, Dijsktra define un semáforo como *un objeto de tipo entero sobre el que se pueden realizar las operaciones: wait (esperar) y signal (incrementar).*

La operación wait decrementa el valor de un semáforo y se utiliza para adquirirlo o bloquearlo mientras que la operación signal incrementará el valor de un semáforo, utilizándose esta operación para liberar un semáforo o inicializarlo.

Sea como fuere, los semáforos nunca pueden tomar valores negativos y si valen 0, no se podrán realizar más operaciones wait sobre él. Vemos que en la mayoría de los casos un semáforo será binario; es decir, sólo valdrá 0 o 1 dependiendo de la situación. Pero es totalmente posible usar semáforos que no sean binarios.

En ocasiones a la operación wait se la denomina operación V (del neerlandés *proberen*, probar), y a la operación signal operación V (del neerlandés *verhogen*, incrementar).

Cuando en programación en UNiX vamos a utilizar semáforos, necesitaremos tener la siguiente información disponible:

- El valor actual del semáforo
- El identificador del proceso que cambió el semáforo por última vez.
- El número de procesos que hay esperando a que el valor del semáforo se incremente.
- El número de procesos que hay esperando a que el semáforo pase a valor 0.

3.3. Programado con semáforos

Tres serán las operaciones básicas relacionadas con semáforos en UNiX:

- Obtención de semáforos
- Acceso a la información de un semáforo
- Manipulación de semáforos (operación)

Estas tres operaciones básicas serán realizadas por tres funciones que veremos a continuación con más detalle: semget, semctl y semop.

3.3.1. Obtención de semáforos (semget)

Una vez más recurrimos a la página del manual para saber cuál es el prototipo de esta función:

```
NOMBRE
    semget - obtiene el identificador de un conjunto de semáforos

SINOPSIS
    # include <sys/types.h>
    # include <sys/ipc.h>
    # include <sys/sem.h>

    int semget ( key_t key, int nsems, int semflg )
```

Como podemos observar, semget nos permite crear o acceder a un conjunto de semáforos bajo un mismo identificador. Si todo va bien, esta función nos devuelve el identificador del conjunto de semáforos; en otro caso retorna -1 (por ejemplo si el conjunto de semáforos asociados a la clave ya existe).

El primer parámetro key es la clave que indica a qué grupo *de semáforos queremos acceder o que grupo de semáforos* queremos crear.

Esta clave se crea mediante ftok:

```
NOMBRE
    ftok - convierte un nombre de camino y un identificador de proyecto en
    una clave IPC de System V

SINOPSIS
    # include <sys/types.h>
    # include <sys/ipc.h>
        key_t ftok ( char *camino, char proy )
```

Como podemos observar esta función nos proporciona una clave IPC a partir de una ruta de fichero y una letra para que podamos utilizar el conjunto de llamadas IPC UNiX.

Estas claves están relacionadas con los inodos que ocupan en disco un determinado archivo, con lo cual el sistema puede generar un mecanismo único para los programas/procesos que compartan recursos IPC como semáforos, memoria compartida, etc.

Todas las funciones relacionadas con la creación de recursos IPC, necesitan de esta clave.

Para que podamos crear semáforos, es imprescindible que el valor de la clave que nos proporcione el sistema sea IPC_PRIVATE.

Si no es así, deberemos de asegurarnos de activar IPC_CREAT en semflg.

Con respectos a los otros parámetros que recibe, nsems será el número de semáforos que habrá en el conjunto de semáforos que vamos a crear semflg indica el modo de creación del semáforo.

Este modo consta de dos partes: el modo IPC que puede valer IPC_CREAT o IPC_EXCL y los permisos que tendrá el semáforo:

Tabla 3. Permisos de creación de semáforos

0400	Permiso de lectura para el usuario.
0200	Permiso de modificación para el usuario.
0060	Permiso de lectura y modificación de grupo
0006	Permiso de lectura y modificación por el resto de usuarios.

Una posible función que podemos construir para crear semáforos es:

```
int crea_semaforo_binario(key_t clave, int semflgs)
{
  return semget(clave, 1, semflgs);
}
```

3.3.2. Accesos a la información del semáforo (semctl)

En varias ocasiones debemos de realizar distintas operaciones sobre los semáforos. La función que emplearemos para ello es **semctl.** a continuación, mostramos toda la página de manual, dado que en ella se nos muestra claramente todas las operaciones que se nos permitirán hacer sobre un conjunto de semáforos:

```
SEMCTL(2)          Manual del Programador de Linux          SEMCTL(2)

NOMBRE
    semctl - operaciones de control de semáforos

SINOPSIS
    #include <sys/types.h>
    #include <sys/ipc.h>
    #include <sys/sem.h>

    #if defined(__GNU_LIBRARY__) && !defined(_SEM_SEMUN_UNDEFINED)
    /* la union semun se define al incluir <sys/sem.h> */
    #else
    /* según X/OPEN tenemos que definirla nosostros mismos */
    union semun {
        int val;                  /* valor para SETVAL */
        struct semid_ds *buf;     /* buffer para IPC_STAT, IPC_SET */
        unsigned short int *array; /* array para GETALL, SETALL */
        struct seminfo *__buf;    /* buffer para IPC_INFO */
    };
    #endif

    int semctl (int semid, int semnun, int cmd, union semun arg)

DESCRIPCIÓN
    La función realiza la operación de control especificada por cmd en el
    conjunto de semáforos (o en el semáforo semnum-avo del grupo) identifi-
    cado por semid. El primer semáforo del conjunto está indicado por el
    valor 0 para semnum.

    Valores válidos para cmd son

    IPC_STAT   Copiar información de la estructura de datos del conjunto de
               semáforos en la estructura apuntada por arg.buf. El argu-
               mento semnum es ignorado. El proceso que realiza la llamada
               debe tener privilegios de acceso de lectura en el conjunto
               de semáforos.

    IPC_SET    Escribir los valores de algunos miembros de la estructura
               semid_ds apuntada por arg.buf a la estructura de datos del
               conjunto de semáforos, actualizando tambien su miembro
               sem_ctime. Los miembros de la estructura provista por el
               usuario struct semid_ds a los que apunta arg.buf son
                   sem_perm.uid
                   sem_perm.gid
                   sem_perm.mode  /* solo los 9 bits más bajos */

               El ID de usuario efectivo del proceso que realiza la llamada
               debe ser o de super-usuario, o el del creador o propietario
               del conjunto de semáforos. El argumento semnum es ignorado.
```

IPC_RMID Borrar inmediatamente el conjunto de semáforos y sus estructuras de datos, despertando todos los procesos en espera (devuelve un error, y errno puesto a EIDRM). El ID de usuario efectivo del proceso que realiza la llamada debe ser o de super-usuario, o el del creador o propietario del conjunto de semáforos. El argumento semnum es ignorado.

GETALL Devuelve semval para todos los semáforos del conjunto, en arg.array. El argumento semnum es ignorado. El proceso que realiza la llamada ha de tener privilegios de lectura en el conjunto de semáforos.

GETNCNT La llamada al sistema devuelve el valor de semncnt para el semnum-avo semáforo del conjunto (p.ej. el número de procesos esperando que aumente semval para el semnum-avo semáforo del conjunto). El proceso que realiza la llamada ha de tener privilegios de lectura en el conjunto de semáforos.

GETPID La llamada al sistema devuelve el valor de sempid para el semnum-avo semáforo del conjunto (p.ej. el pid del proceso que ejecutó la última llamada semop para el semnum-avo semáforo del conjunto). El proceso que realiza la llamada ha de tener privilegios de lectura en el conjunto de semáforos.

GETVAL La llamada al sistema devuelve el valor de semval para el semnum-avo semáforo del conjunto. El proceso que realiza la llamada ha de tener privilegios de lectura en el conjunto de semáforos.

GETZCNT La llamada al sistema devuelve el valor de semzcnt para el semnum-avo semáforo del conjunto (p.ej. el número de procesos que esperan que semval del semnum-avo semáforo del conjunto se ponga a 0). El proceso que realiza la llamada ha de tener privilegios de lectura en el conjunto de semáforos.

SETALL Poner semval para todos los semáforos del conjunto usando arg.array, actualizando también el miembro sem_ctime de la estructura semid_ds asociada al conjunto. Los registros de anulación son limpiados, para los semáforos cambiados, en todos los procesos. Los procesos que están durmiendo en la cola de espera son despertados si algún semval se pone a 0 o se incrementa. El argumento semnum es ignorado. El proceso que realiza la llamada ha de tener privilegios de modificación en el conjunto de semáforos.

SETVAL Poner el valor de semval a arg.val para el semnum-avo semáforo del conjunto, actualizando también el miembro sem_ctime de la estructura semid_ds asociada al conjunto. El registro de anulación es limpiado, para el semáforo alterado, en todos los procesos. Los procesos que están durmiendo en la cola de espera son despertados si semval se pone a 0 o se incrementa. El proceso que realiza la llamada ha de tener privilegios de escritura en el conjunto de semáforos.

VALOR DEVUELTO

Si falla, la llamada al sistema devuelve -1, mientras errno indica el error. De otro modo, la llamada al sistema devuelve un valor no negativo, dependiendo de cmd como sigue:

GETNCNT el valor de semncnt.

GETPID el valor de sempid.

GETVAL el valor de semval.

GETZCNT el valor de semzcnt.

ERRORES

En caso de error, errno tendrá uno de los siguientes valores:

EACCESS El proceso que reliza la llamada no tiene los permisos necesarios para ejecutar cmd.

EFAULT La dirección a la que apunta arg.buf o arg.array no es accesible.

EIDRM El conjunto de semáforos se borró.

EINVAL Valor no válido para cmd o semid.

EPERM El argumento cmd tiene valor IPC_SET o IPC_RMID pero el user-ID del proceso que realiza la llamada no tiene suficientes privilegios para ejecutar el comando.

ERANGE El argumento cmd tiene el valor SETALL o SETVAL y el valor al que ha de ser puesto semval (para algún semáforo del conjunto) es menor que 0 o mayor que el valor SEMVMX de la implementación.

NOTAS

Las llamadas de control IPC_INFO, SEM_STAT y SEM_INFO son utilizadas por el programa ipcs(8) para proveer información sobre recursos asignados. En el futuro pueden ser modificadas según se necesite, o llevadas al interfaz del sistema de ficheros proc.

El siguiente limite de sistema para conjuntos de semáforos afecta a lallamada semctl:

SEMVMX Valor máximo para semval: depende de la implementación (32767).

CONFORME A

SVr4, SVID. SVr4 documenta adicionalmente las condiciones de error EINVAL y EOVERFLOW.

VÉASE TAMBIÉN

ipc(5), shmget(2), shmat(2), shmdt(2)

Linux 0.99.13 1 noviembre 1993 SEMCTL(2)

De esta página, sacamos la conclusión de que nosotros deberemos de declarar manualmente la unión **semun** que nos permitirá acceder y manipular la información asociada a un conjunto de semáforos:

```
union semun {
  int val;                    /* valor para SETVAL */
  struct semid_ds *buf;       /* buffer para IPC_STAT, IPC_SET */
  unsigned short int *array;  /* array para GETALL, SETALL */
  struct seminfo *__buf;      /* buffer para IPC_INFO */
};
```

Un ejemplo de uso, sería la creación de una función que destruyese el conjunto de semáforos creados, y otra que los inicializase:

```
//
// Función que destruye un conjunto de semáforos
//
int eliminar_semaforo_binario(int semid)
{
 union semun argumento_tonto;
 return semctl(semid, 1, IPC_RMID, argumento_tonto);
}

//
// Función que inicializa un conjunto de semáforos
//
int inicializa_semaforo_binario(int semid)
{
 union semun argumentos;
 unsigned short valores[1];

 valores[0]=1; // Inicializamos el semáforo a 1
 argumentos.array = valores;
 return semctl(semid, 0, SETALL, argument);
}
```

3.3.3. Operaciones sobre semáforos (semop)

Una vez creados los semáforos e inicializados, sobre ellos podremos ejecutar las órdenes **wait** y **signal** . El prototipo de esta función es:

```
# include <sys/types.h>

# include <sys/ipc.h>

# include <sys/sem.h>

int semop ( int semid, struct sembuf *sops, unsigned nsops )
```

Siendo **semid** el identificador del conjunto de semáforos sobre el que vamos a operar, **sops** las operaciones a realizar y **nsops** el número de operaciones a realizar. **sops** es un puntero a un array de operaciones (de tantos elementos como se indique en **nsops**) teniendo cada elemento el siguiente aspecto:

```
struct sembuf{
        short sem_num;  /* numero de semaforo: 0 = primero */
        short sem_op;   /* operacion sobre el semaforo */
        short sem_flg;  /* banderas (indicadores/parametros) de la operacion */
};
```

Aquí, **sem_num** indica el número del semáforo sobre el que vamos a operar dentro del conjunto de semáforos.

Con respecto a **sem_op** es la operación a realizar sobre este semáforo.

Si esta variable es negativa, se realizará una operación de decremento sobre el semáforo (operación **wait**), si es positivo, se incrementará dicho valor (operación **signal** y si vale **0** no se realizará operación alguna.

Finalmente, **sem_flg** es un conjunto de banderas que dependiendo de su valor (**IPC_NOWAIT** o **SEM_UNDO**) nos indicará que ocurrirá si la operación no tiene éxito (en el primer caso se retorna el control, en el segundo se deshace la operación realizada cuando el proceso termina).

3.3.4. Para terminar, un ejemplo

Soy consciente de que a pesar del gran volumen de información suministrada, de poco nos servirá si no la entendemos. Debido a la extensión de este artículo, me gustaría haber explicado mejor el tema de semáforos. Confío sin embargo que un ejemplo aclare el funcionamiento de los semáforos. Procuraré indicar en el código todo lo necesario.

```
/////////////////////////////////////////////////////////////////////////////////////
// ARCHIVO: semaforo.c
//
// DESCRIPCIÓN:
//   Muestra el empleo de un semáforo binario.
//
//
// AUTOR:
//   Luis U. Rodríguez Paniagua
//   Este código se distribuye bajo los términos de la
//   Licencia Pública General (GPL) de GNU. Puede usted
//   consultarla en www.gnu.org
//
/////////////////////////////////////////////////////////////////////////////////////
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <stdio.h>
//
//
// Definimos unas constantes

#define WAIT  -1
#define SIGNAL  1
#define NUM_SEMAFOROS 2 // Dos procesos, dos semáforos
#define PADRE   0
#define HIJO    1
//
// Definimos la unión semun
//
union semun {
  int val;                    /* valor para SETVAL */
  struct semid_ds *buf;       /* buffer para IPC_STAT, IPC_SET */
  unsigned short int *array;  /* array para GETALL, SETALL */
  struct seminfo *__buf;      /* buffer para IPC_INFO */
};
//
// Función que crea un conjunto de semáforos
//
int crea_semaforos(key_t clave, int semflgs)
{
      return semget(clave, NUM_SEMAFOROS , semflgs);
}
//
// Función que destruye un conjunto de semáforos
//
int eliminar_semaforos(int semid)
 {
   union semun argumento_tonto;
   return semctl(semid, 0, IPC_RMID, argumento_tonto);
 }


//
// Función que inicializa un conjunto de semáforos
//
int inicializa_semaforos(int semid, int semnum, int valor)
{
  union semun argumentos;
  unsigned short valores[2];

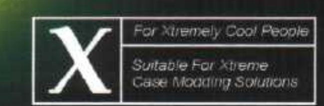
  valores[semnum]=valor; // Inicializamos el semáforo a 1
  argumentos.array = valores;
  return semctl(semid, 0, SETALL, argumentos);
}
//
// Función que realiza la operación WAIT
//
int operacion_wait(int semid, int semnum)
{
      struct sembuf operacion[2];
      operacion[semnum].sem_num = semnum;
      operacion[semnum].sem_op  = WAIT;
      operacion[semnum].sem_flg = SEM_UNDO;
      return  semop(semid, operacion, NUM_SEMAFOROS);
}
//
// Función que realiza la operación SIGNAL
//
int operacion_signal(int semid, int semnum)
{
     struct sembuf operacion[2];
     operacion[semnum].sem_num = semnum;
     operacion[semnum].sem_op  = SIGNAL;
     operacion[semnum].sem_flg = SEM_UNDO;
     return  semop(semid, operacion, NUM_SEMAFOROS);
}
```



"Be Cool! Be Aerocool!"



RAFAEL ALBERTI, 24 BAJO
01010 VITORIA-GASTEIZ (SPAIN)
TEL.: 902-227733 / 945-176647
FAX.: 94-4342433
http://www.biomag.biz
E-MAIL: compras@biomag.biz







¡La nueva generación de cajas de ordenador!





¿Tiene un ordenador siempre que parecer un ordenador?
¡¡ NO !! ¡¡ NO MAS MONOTONIA!!!

¡¡Ahora, transforma tu ordenador "cuadrado" en cualquier cosa!! Presentamos las nuevas cajas "LUBIC". Gracias al concepto modular de "LUBIC", dispones de todo lo necesario para crear tu propio diseño de caja de ordenador con los elementos de aluminio incluidos en el producto

Planifica cuidadosamente el plan de construcción y los materiales para transformar tu ordenador en algo más que una simple y aburrida caja de metal.
¡¡Todo es posible con las cajas "LUBIC" !! Puedes crear una caja "Escorpión", una caja "Elefante", una caja "bombardero B-52" o cualquier cosa que imagines. También puedes crear cosas que no sean ordenadores con los modulos "LUBIC" como portaretratos, expositores o mesas.

¿Es el cielo el limite?
Con los módulos "LUBIC", ¡¡NO EXISTEN LIMITES!!!
¡¡¡Da forma a todas sus ideas!!! ¡¡Las creaciones solo estan limitadas por tu imaginación!!!



Los paquetes estandar son:

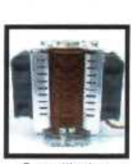
1. LUBIC-3519-BL
2. LUBIC-3519-BK
3. LUBIC-3519-SV
4. LUBIC-4480-BL
5. LUBIC-4480-BK
6. LUBIC-4480-SV
7. LUBIC-AIRPLANE-BL
8. LUBIC-AIRPLANE-BK
9. LUBIC-AIRPLANE-SV

Serie Deep Impact
DP-102




Maximo Diseño


1 ventilador


2 ventiladores

Caracteristicas

1. Alta conductividad térmica
2. Totalmente fabricado en cobre con un tubo superconductor para un maximo rendimiento
3. Super Diseño circular
4. DP-102 ofrece una solución de doble ventilador - pueden usarse uno o dos ventiladores. (Ventiladores no incluidos en el paquete)
5. Incluye dos sets de adaptadores de ventilador de 70mm a 80mm.
6. El disipador puede ser rotado 360 grados y los ventiladores pueden ser instalados en cualquier posición.
7. DP-102 puede ser "Universal" compatible con AMD e Intel P4

Aplicaciones

AMD: Athlon XP 3600+ y superiores
Intel: P4 Socket 478 3.6Ghz y superiores (solo aplicable a la versión "Universal", no disponible para la versión "AMD")

Disipador

Dimensiones =Tubo -100 mm, 36+ láminas - dia. 66mm
Material =Tubo Superconductor + Láminas de Cobre

```
//
// Función que nos devuelve el valor de un semáforo
//
int obten_valor_semaforo(int semid, int num_sem)
{
    union semun argumento_tonto;
    return semctl(semid, num_sem, GETVAL, argumento_tonto);
}
int main(int argc, char *argv[])
{
    int semaforo, hijo, i, j;
    key_t clave;
    // Creamos la clave a partir del nombre del propio ejecutable
    // y la letra L
    clave = ftok(argv[0],'L');
    // Creamos el cjoto de semáforos
    if (( semaforo = crea_semaforos(clave, IPC_CREAT | 0600) ) == -1)
    {
        printf("Error al crear el semáforo.\n");
        return -1;
    }
    // Inicializamos el semáforo del padre
    inicializa_semaforos(semaforo, PADRE, 1);
    // Inicializamos el semáforo del hijo
    inicializa_semaforos(semaforo, HIJO, 1);
    // Creamos el proceso hijo.
    if ( (hijo = fork()) == -1)
    {
        printf("Error al crear proceso hijo.\n");
        return -1;
    }
    if(hijo == 0)
    {
        // Si somos el hijo
        for(i=0;i<100;i++)
        {
            // Bloqueamos el proceso hijo
            operacion_wait(semaforo, HIJO);
            // Realizamos nuestro trabajo
            fprintf(stderr,"o");
            // Desbloqueamos el proceso padre
            operacion_signal(semaforo, PADRE);
        }
        eliminar_semaforos(semaforo);
    }
    else
    {
        // Somos el padre
        for(j=0;j<100;j++)
        {
            // Bloqueamos el proceso padre
            operacion_wait(semaforo, PADRE);
            // Hacemos nuestro trabajo
            fprintf(stderr,"+");
            // desbloqueamos al proceso hijo
            operacion_signal(semaforo, HIJO);
        }
        eliminar_semaforos(semaforo);
    }
    return 0;
}
```

Este programa deberá imprimir los símbolos o y + alternativamente.

Bibliografía

Fco. Manuel Márquez, *UNIX Programación Avanzada*, 2ª Edición, Ra-Ma, [MÁRQUEZ].

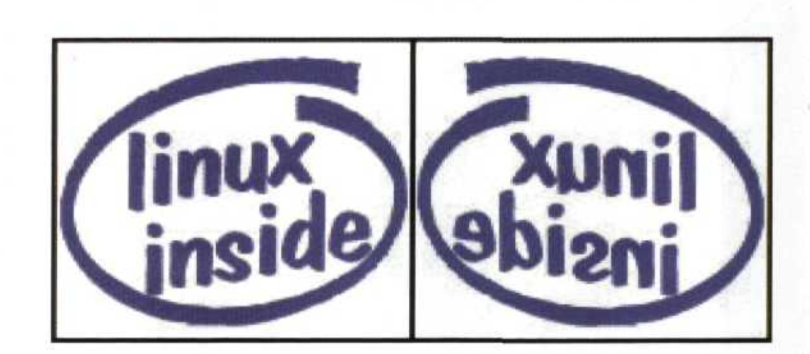
Mark Mitchell, Jeffrey Oldham, Alex Samuel, *Advanced LiNUX Programming*, First Edition, New Riders, [MARK].

Antonio P. Giménez Lorenzo, *UNIX System V*, 1ª Edición, Anaya Multimedia, [GIMÉNEZ].

Jorge E. Pérez Martínez, *Programación Concurrente*, 2ª Edición corregida, Editorial Rueda, [PÉREZ].

Juan M. Morera Pascual, Juan A. Pérez-Campanero Atanasio, *Teoría y diseño de Sistemas Operativos*, 1ª Edición, Anaya Multimedia, [MORERA].

Páginas del manual UNiX, Varios, [MANUAL].



CURSO DE TCP/IP
INTRODUCCION

Si alguna vez has intentado comprender Internet, seguro que has acabado frente a un libro de TCP-IP. Y seguro que a la sexta página lo has dado por IMPOSIBLE!!! TCP-IP es el alma de la red, nosotros te ofrecemos un curso MUY ESPECIAL ;)

1.Introducción a la introducción.

Probablemente muchos de vosotros esperabais con impaciencia un curso como este, así que espero que vuestra alegría, al ver que al fin nos metemos de lleno con el tema en esta revista, no se vea frustrada al descubrir que vuestro "profe" en este curso va a ser el mismo que mes tras mes os ha ido aburriendo soberanamente con la serie RAW. ;-)

Si es cierto que os parezco aburrido, en mi defensa he de alegar que la descripción detallada de un protocolo es algo aburrido de por sí y, aunque he hecho lo posible por hacer artículos amenos, cuando hay que ponerse serio, hay que ponerse serio.

Aprovecho esta ocasión para agradecer a Adhara (conocida como MariAn antes de digievolucionar) su aportación en este sentido, con las caricaturas e ilustraciones que ha aportado para amenizar en la medida de lo posible la serie RAW y que, por supuesto, también seguirá aportando en este curso que comienza.

Os plantearé el terrible dilema que he sufrido para poder comenzarlo. Para ayudarme a la hora de estructurar un poco las ideas he ojeado multitud de libros y cursos de TCP/IP para ver cómo abordaban el problema y poder explicarlo desde cero, que era mi gran reto.

Lamentablemente, en ninguno he encontrado lo que yo buscaba, una forma de introducir los conceptos de forma que alguien sin ningún conocimiento pueda no sólo aprenderse de memoria un montón de formatos de cabeceras, comandos de protocolos, y un glosario de términos; si no realmente llegar a hacer suyos los conceptos y comprenderlos de una forma totalmente natural.

Por supuesto, ya se que la mayoría de los lectores de esta revista tienen ya un nivel aceptable de conocimientos, pero he de enfocarlo no sólo para los lectores "veteranos", si no también para aquellos totalmente novatos que, sin duda, serán los que más se beneficien de este curso y que les abrirá las puertas para llegar a comprender en profundidad todo lo demás que se enseñe en la revista a partir de ahora.

Sin duda alguna, el TCP/IP es uno de los pilares de todo este jaleo en el que estamos metidos. ;-) Así que decidí coger al toro por los cuernos, y enfocar la cuestión desde un punto de vista diferente.



TCP/IP

TCP / IP: Transmission Control Protocol / Internet Protocol = Protocolo de Control de Transmisión / Protocolo de Internet

¿Cuántos cursos de TCP/IP empiezan contando el modelo **OSI (Open Systems Interconnection = Interconexión de Sistemas Abiertos)**? A los que hayáis seguido alguno de esos cursos publicados en Internet o en otras revistas, ¿os quedó claro desde el principio, por ejemplo, para qué servia la capa de sesión del modelo OSI? ¿No pensáis que quizá empezar abordando el problema planteando un modelo teórico tan complejo puede ser contraproducente? ¿No os quedaron

más dudas después de terminar el tema de introducción que antes de empezarlo? Seguro que la mayoría dejasteis el curso a la mitad (y otros ni siquiera pasasteis de las primeras páginas).

El empezar cualquier curso de TCP/IP hablando del modelo OSI parece que ha sido la solución estándar para solucionar el reto de mostrar el concepto de los protocolos por capas a gente totalmente nueva en el tema. Pero a mí personalmente nunca me ha parecido una buena idea, así que he tomado una medida muy arriesgada, y es intentar dar un nuevo enfoque a este reto.

No sé qué resultados tendrá mi enfoque, pero espero que al menos sea una alternativa para que aquellos que no terminan de captar los conceptos por los medios "clásicos" tengan aquí una segunda oportunidad.

¿Qué esperabais? ¿Que mi curso de TCP/IP fuese como todos los demás? ¡Para eso tenéis millones de libros sobre el tema! Lo que pretendemos dar en esta revista son nuevas visiones que no se pueden encontrar en las fuentes "convencionales".

El juzgar si nuestro enfoque es mejor que el convencional, ya depende de cada lector, y confío en que todos vosotros tendréis buen juicio para escoger la opción que para cada uno de vosotros resulte más adecuada.

Como veréis, mi enfoque intenta mostrar los conceptos puros, al margen de cualquier detalle técnico, mediante varios símiles con otros conceptos, bien conocidos por todos, de nuestra vida cotidiana.

Por supuesto, después de este primer artículo vendrán otros, y ahí sí que veremos ya en profundidad los detalles técnicos, los cuales nos entrarán con muchísima más facilidad si previamente hemos dedicado un tiempo imprescindible a llegar al fondo de los conceptos.

2. El concepto fundamental de protocolo por capas

Empezaremos nuestro viaje metafórico por el mundo de los protocolos situándonos en un hospital, donde los doctores PyC y Scherzo hablan acerca de la próxima operación a corazón abierto que tendrán que llevar a cabo.

El doctor PyC tiene unas dudas acerca de la complicadísima operación, y acude al doctor Scherzo en busca de ayuda.







Dios mío, como haya entre los lectores algún médico o estudiante de medicina... ¡que me perdone por la increíble cantidad de estupideces que acabo de soltar! XD

Al margen de si tiene sentido o no la parrafada, supondremos que se trataba de una conversación perfectamente normal entre cirujanos.

Ahora vamos a plantearnos una serie de cuestiones sobre estas viñetas. En primer lugar, ¿qué han hecho básicamente PyC y Scherzo?
COMUNICARSE.

Ahora vamos a analizar un poco en qué ha consistido esa comunicación.

En primer lugar, lo que más nos llama la atención es el lenguaje técnico utilizado, que sólo es comprendido por los cirujanos.

Igual que los cirujanos tienen su propio lenguaje técnico, los informáticos también tienen el suyo, los arquitectos el suyo, y los abogados el suyo, todos ellos diferentes entre sí.

Pero, a pesar de que todos estos lenguajes técnicos sean diferentes, todos ellos se apoyan en una misma base, que es el idioma; en este caso, el castellano.

El lenguaje técnico de los cirujanos consiste únicamente en una serie de palabras y expresiones que permiten expresar los términos específicos que requieren los cirujanos para comunicarse en su trabajo. Por tanto, no es un lenguaje completo, ya que no posee una gramática propia que permita mantener una conversación sin apoyarse en un idioma básico, como puede ser el castellano.

Si, por ejemplo, eliminásemos de la parrafada del doctor PyC todo aquello que no formase parte exclusivamente del lenguaje técnico de los cirujanos, esta frase:



Se convertiría en ésta otra:



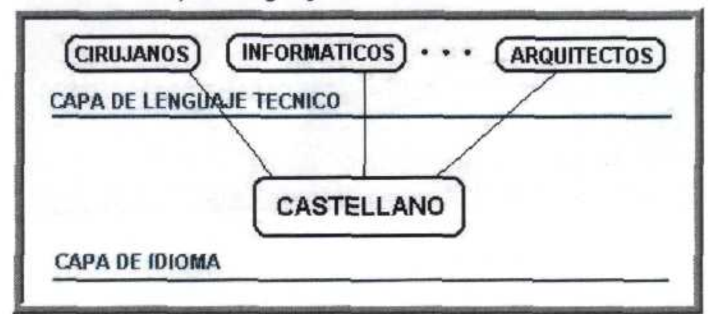
Por la cara que pone el doctor Scherzo podemos estar seguros de que utilizando tan sólo el lenguaje técnico, sin apoyarnos en la base que es el idioma castellano, es imposible que dos cirujanos se comuniquen.

Lo mismo que pasa con los cirujanos pasa con cualquier otro grupo de profesionales que utilicen su propio lenguaje técnico. Todos ellos apoyan toda su conversación en un idioma común, que puede ser el castellano, el inglés, o cualquier otro.

Por supuesto, para que dos profesionales se entiendan tienen que hablar no sólo el mismo lenguaje técnico, si no también el mismo idioma común.

Si el doctor PyC hablase Japonés, sin duda el doctor Scherzo habría puesto la misma cara de incomprensión.

Según lo que hemos visto hasta ahora, la comunicación entre los dos doctores funciona gracias a dos capas independientes: el idioma, y el lenguaje técnico.



¿Cuál es el motivo por el cual es esto así? Pues, si pensáis un poco, llegaréis vosotros *mismos a la conclusión.*

Imaginad que el lenguaje técnico de los cirujanos fuese un lenguaje completo, con sus fórmulas de saludos, despedidas, una gramática completa para construir las frases, palabras para expresar cualquier término común en cualquier comunicación (como los *habituales: "me lo repita", "¡habla más* despacio, que no me da tiempo a apuntarlo!", etc.), e incluso tuviese sus propios nombres, en lugar de los que tenemos en castellano (doctor Pyc, y doctor Scherzo). ¡Sería una completa locura!

Desde luego, no sería nada práctico que *cualquier cirujano tuviese que aprender un* idioma totalmente nuevo sólo para poder comunicarse con sus colegas.

Lo más práctico, y lo más lógico, es utilizar el recurso conocido por todos que es el idioma castellano, y simplemente ampliarlo con una serie de términos que permitan entrar en detalle en los conceptos manejados por los cirujanos.

Una vez comprendida la necesidad de comunicarse utilizando dos capas, vamos a entrar más en profundidad en la comunicación que vimos en las viñetas.

¿Qué otro medio común han utilizado el doctor *Scherzo y el doctor Pyc para comunicarse?* ¡El habla!

Si trasladásemos toda esa conversación a un papel, ¿no tendría el mismo sentido? ¿Y si la trasladásemos a una conversación telefónica? ¿O a una conversación por IRC (Internet Relay Chat)?

Los dos doctores se han apoyado en un medio físico común, que es la voz, pero perfectamente podrían haber mantenido la misma conversación a través de otro medio físico, como la escritura, o el teléfono.

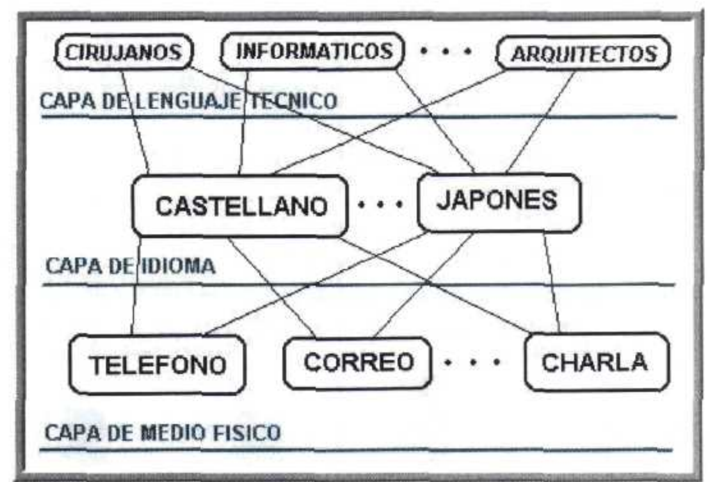
Tanto si esa conversación es hablada como si es escrita, seguiría utilizando tanto el lenguaje *técnico de los cirujanos, como el idioma* castellano. En nada cambiaría, salvo en el hecho de que el medio utilizado sería diferente.

Ahora bien, igual que un cirujano japonés no puede entenderse con un cirujano de Jaén, si el doctor PyC le hubiese soltado la parrafada al doctor Scherzo por correo, y éste le hubiese *respondido a viva voz cuando recibiese la* carta (es decir, que se lo habría contado a las paredes), tampoco habría sido posible una comunicación.

Ambos interlocutores tienen que compartir el mismo medio físico para comunicarse. Si un interlocutor está utilizando el teléfono, y el otro está respondiendo por escrito en un papel, jamás podrá haber una comunicación.

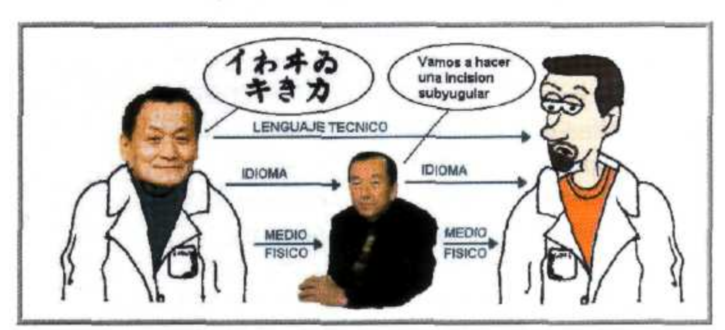
Por supuesto, tampoco sirve de nada que ambos hablen a viva voz, si cada uno está en un lugar diferente, donde no se puedan escuchar mutuamente.

Podemos considerar, por tanto, al medio físico como otra capa de la comunicación. En este caso, esta capa ya no existe por una conveniencia de hacer las cosas más fáciles, si no por una necesidad natural.



Vamos a ver qué ocurre ahora si el doctor Me-Iwa, de Japón, quiere hablar con el doctor PyC acerca de la operación de cardiopatía precarótida.

Por supuesto, no podrán comunicarse directamente, al hablar distintos idiomas, pero por suerte el doctor Me-Iwa tiene un intérprete que puede hablar tanto en castellano como en japonés, por lo que éste sería el escenario ahora:



Ahora meditemos un poco acerca de este nuevo personaje, el intérprete. Lo más probable es que este intérprete sea un estudiante de filología, o simplemente un estudiante de idiomas de academia pero, en cualquier caso, es poco probable que el *intérprete sea un cirujano.*

Pero, ¿es realmente necesario que el intérprete sepa algo de cirugía? El lenguaje técnico de los cirujanos al fin y al cabo no es más que una extensión del idioma, por lo que bastaría con que el intérprete simplemente conociese ambos idiomas para transmitir la conversación entre los dos interlocutores. Nos da exactamente igual que el intérprete no entienda ni papa de la conversación, ya que esa conversación no va dirigida a él, no es más que un mero intermediario. Él tiene que saber únicamente ambos idiomas: japonés y castellano.

Una vez que el intérprete ha traducido la frase "Vamos a hacer una incisión subyugular" ya será problema de los cirujanos entenderse entre ellos, ya que el intérprete no tiene ni idea de qué es una incisión subyugular, pero si que sabe traducir las palabras literalmente.

Por tanto, podríamos considerar al intérprete como un intermediario en la conversación que sólo llega a comprender hasta cierta capa de la comunicación pero que, aún así, es capaz de transmitir todo, confiando en que los interlocutores serán capaces de comprenderse *una vez traducido el idioma.*

Más adelante profundizaremos mucho más en la cuestión de los intermediarios en la comunicación, así que quedaos bien con esta idea. ;-)

3. Las capas de TCP/IP

¡Al fin vamos a ver la relación que tiene todo esto con el tema que nos interesa! Ya hemos comprendido la necesidad de estructurar la comunicación en diferentes capas, tanto por necesidad física, como por conveniencia para facilitar la comunicación (reducir su complejidad). Por supuesto, eso es algo que ocurre en todo tipo de comunicación y, por tanto, la comunicación entre máquinas no va a ser menos.

Por un momento, imaginemos que no hubiese capas en la comunicación por Internet.

Si yo tuviese que programar un cliente de correo electrónico (como por ejemplo el Outlook Express, tendría que idear desde cero toda una serie de funciones para interconectar al remitente y al destinatario, una serie de

funciones para asegurarme de que el mensaje llegue sin errores hasta el destino, una serie de funciones para identificar a ambas partes, etc., etc.

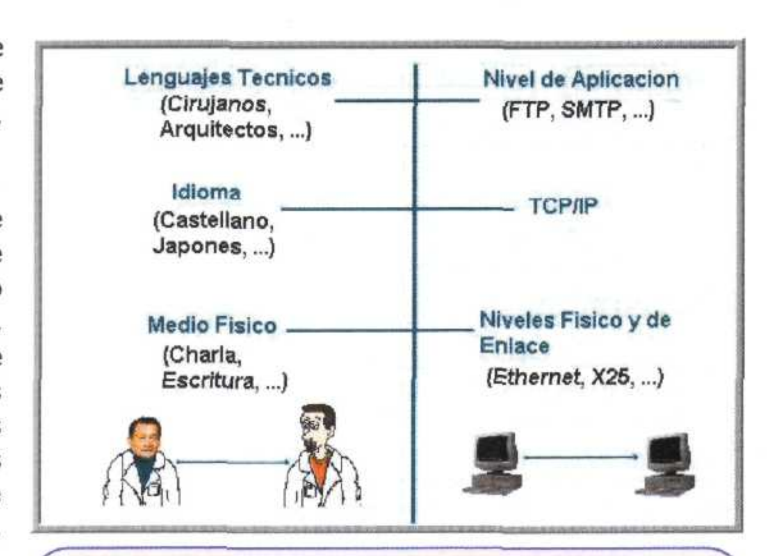
Si después me da por programar un cliente de FTP (si no sabes lo que es el FTP -File Transfer Protocol-, descárgate gratis el número 1 de esta revista desde www.hackxcrack.com), tendría que inventarme de nuevo y desde cero unas funciones para interconectar ambas partes (nuestro cliente con el servidor), unas funciones para asegurarme de que los ficheros y los comandos lleguen sin errores, una serie de funciones para identificar ambas partes, etc., etc.

El hacer las cosas de esta manera, no sólo sería una cantidad ingente de trabajo innecesario, si no que además dificultaría enormemente que los programas se entendiesen entre sí.

Si todas las aplicaciones que utilizan Internet tienen que realizar una serie de tareas comunes, como son la interconexión de las partes implicadas en la comunicación, la identificación de ambas partes, la corrección de errores, el ajuste de las velocidades de recepción y transmisión, etc., etc., ¿por qué no utilizar un lenguaje común para todas ellas?

Igual que todos los profesionales (cirujanos, informáticos, arquitectos...) utilizan el idioma castellano como base sobre la cual apoyan luego sus lenguajes técnicos propios, también las máquinas conectadas a Internet utilizan un mismo idioma común como base sobre la que luego apoyar cada lenguaje específico.

En este caso, el idioma común de las máquinas es el famoso TCP/IP, y los lenguajes técnicos que utiliza cada máquina apoyándose en TCP/IP son los que permiten las diferentes tareas, como transmitir ficheros (FTP), enviar correo (SMTP), mostrar páginas Web (HTTP), etc., etc.



Comparación entre las capas de la comunicación entre dos personas y la comunicación entre dos máquinas. Esta comparación no es muy precisa, así que la muestro sólo como una primera aproximación a la idea.

Los que hayáis seguido mi serie RAW desde el principio, comprenderéis ahora por qué una y otra vez repetía frases como: "protocolo que funciona sobre TCP/IP", o "protocolos por encima del nivel de TCP/IP", etc.

Por ejemplo, ¿tenéis a mano el número 14 de la revista, en el que hablaba sobre el protocolo **DNS**? Mirad el primer párrafo de ese artículo, y veréis que decía:

"Por primera vez en la ya veterana serie RAW, no se trata de un protocolo basado en TCP, ¡si no en el aún desconocido UDP!"

Esto sería lo mismo que decir: "este lenguaje que vamos a contar no se basa en el idioma castellano, si no en el idioma japonés".

Ahora ya comprendemos la necesidad de separar por capas las comunicaciones entre máquinas para facilitar la comunicación, y reutilizar el trabajo ya hecho para no tener que reprogramarlo cada vez.

Y precisamente esas funciones que son utilizadas por muchos programas para que estos no tengan que reprogramarlas desde cero, son precisamente las funciones que te da la API de un **Sistema Operativo**.

Por ejemplo, un programa que funcione bajo Windows no tiene que preocuparse de saber cómo dibujar una ventana en la pantalla, si no que simplemente le dice al sistema "dibújame una ventana de estas características" y Windows hará el trabajo sucio por él.

Todavía recuerdo los tiempos en los que programaba aplicaciones gráficas en MS-DOS y me tenía que currar desde cero todo el interfaz... un auténtico infierno. Perdías más tiempo con el interfaz que con el propio programa.

Pues lo mismo que ocurre con las ventanas, que son una función común a todas las aplicaciones de Windows, también ocurre con las comunicaciones, que tienen una serie de funciones comunes a todas las aplicaciones de comunicaciones. Estas funciones comunes, que son las que proporciona el "idioma" TCP/IP, se ubican precisamente en el Sistema Operativo, para que sea él el que lidie con los detalles, *igual que las ventanas las gestiona* el Sistema Operativo, y es él el único que se preocupa de conocer los detalles para dibujarlas.

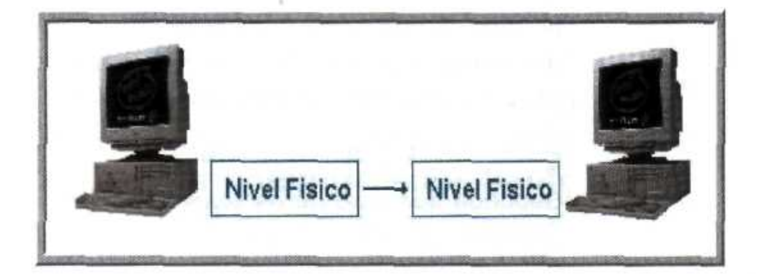
Configuración de TCP/IP en Windows.

Este es el motivo por el que, antes de conectar con Internet o con cualquier otra red, tenemos que configurar el protocolo TCP/IP en nuestro sistema.

A lo largo del curso probablemente veremos cómo configurar correctamente el "idioma" TCP/IP con diferentes sistemas.

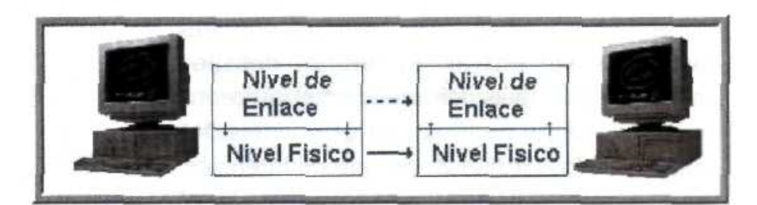
Por el momento, continuaremos con los conceptos sin entrar en ningún detalle. Ahora que ya habéis comprendido el concepto de capas, he de pediros que os olvidéis del *ejemplo de los cirujanos, porque mi intención* era únicamente que comprendieseis el concepto de capas, pero no mostrar metafóricamente cada capa del protocolo TCP/IP con su equivalente en el "mundo real", ya que las capas que forman TCP/IP no tienen prácticamente nada que ver con las capas que forman la comunicación entre dos cirujanos.

La única capa que sí que tienen en común tanto las máquinas como los cirujanos es la del **medio físico** ya que, como dije, esta capa no surge como una facilidad para la comunicación, si no que es una necesidad natural irremediable. Igual que dos cirujanos necesitan compartir un mismo medio para comunicarse, también han de hacerlo dos máquinas.



En el caso de las máquinas, hay que tener en cuenta no sólo la tecnología utilizada en los *propios "cables" que interconectan las* máquinas (que sería el medio físico) si no también el cómo están conectadas: cada cable conecta sólo dos máquinas, un sólo cable conecta a la vez a muchas máquinas entre sí, etc. Es decir, cómo se **enlazan** las máquinas entre sí. Ya veremos mucho más sobre esto más adelante, pero de momento nos quedaremos con la idea de que existe una segunda capa, conocida como **capa de enlace**.

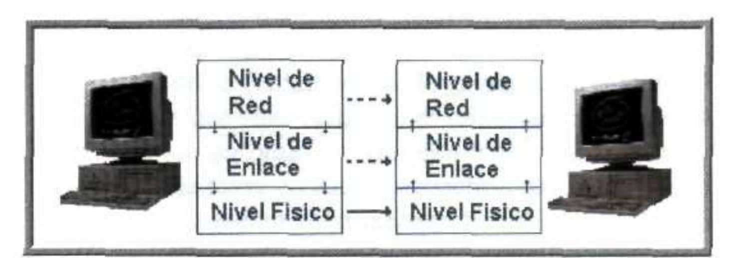
Para los impacientes, deciros que aquí es donde se ubicaría el famoso protocolo **ARP (Protocolo de Resolución de Direcciones)**, que es una parte de la capa de enlace utilizada normalmente en nuestros PCs.



Por encima del nivel de enlace, tenemos ya *la primera capa que realmente nos sonará*, que es la llamada **capa de red** y que, en nuestro caso (el caso de TCP/IP) es la capa llamada **IP**.

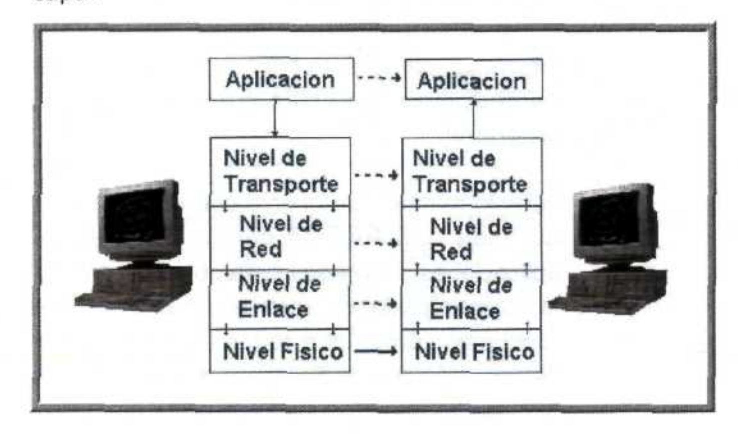
¿Cuál es la responsabilidad de esta capa? Pues principalmente una: conseguir que la comunicación llegue desde el origen hasta el destino. *Ésta es la capa encargada* de identificar a las diferentes máquinas, para que éstas puedan diferenciarse unas de otras, y mantener así comunicaciones separadas. Si no existiese esta capa, todos los datos de Internet llegarían a todas las máquinas conectadas a la red. No habría forma de ver una página Web, ya que no se podría diferenciar una de otra; no se podría enviar un e-mail, ya que no sabríamos cómo encontrar nuestro servidor de correo, etc., etc.

Supongo que esto no os sonará muy raro, teniendo en cuenta que las direcciones utilizadas en Internet se llaman precisamente **direcciones IP**. ¿Casualidad? ;-)



Pero, aún nos faltan muchas funciones comunes, ¿verdad? Como podéis adivinar, del resto de funciones se encarga la capa que está por encima de la capa IP que es, *precisamente, la capa* **TCP**.

En cualquier modelo (ya que TCP/IP no es el único modelo de protocolo por capas que existe) a esta capa se la conoce como **capa de transporte**. Las funciones de esta capa son muchas, pero básicamente podríamos resumirlo en una: permitir el establecimiento de conexiones independientes y seguras, con todo lo que ello implica. Para comprender esto, así como para comprender mejor la capa IP, os muestro a continuación una serie de puntos que, de nuevo a base de símiles, os explicarán muchas de las funciones llevadas a cabo por cada capa.



3.1. La capa IP: La necesidad de direccional

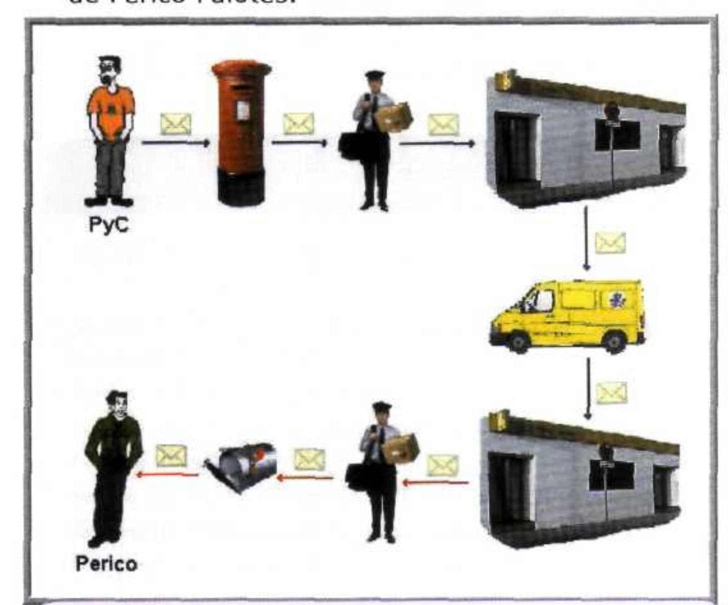
Para comprender la necesidad de la capa IP (Internet Protocol = Protocolo de Internet) presentaremos un nuevo escenario, bien conocido por todos, que es el **correo postal** de toda la vida.

En nuestro ejemplo, PyC, residente en Madrid, quiere enviar una carta a la siguiente dirección:

Perico Palotes
C/Pirulin. Nº12. 1º A.
35003 - Villapalotes (Huelva).

¿Cómo puede llegar hasta su destino la carta después de que PyC la eche en el buzón de su barrio? Todos conocemos bien el proceso. Primero, el cartero del barrio recoge todas las cartas del buzón, y las lleva a la oficina de correos más cercana. Una vez ahí, se ve que el destino de esta carta es Huelva y, por tanto, el sobre es transportado hasta esa provincia en un furgón de Correos. En esa provincia, el sobre se lleva a la oficina de correos

correspondiente. En esa oficina, comprueban la dirección, y llevan la carta hasta el buzón personal de Perico Palotes.



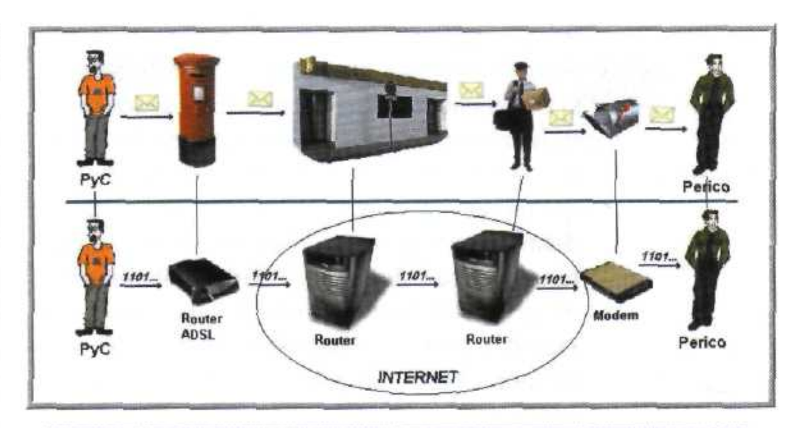
Funcionamiento básico del sistema de correo postal.

Gracias a las direcciones postales todo el sistema de Correos puede funcionar. Y no sólo gracias a la dirección del destinatario, si no también a la del remitente, que será una dirección con el mismo formato (nombre, calle, código postal, población, y provincia).

Gracias a la dirección del remitente se sabrá a quién informar si la carta no llega a su destino, y el destinatario podrá responder a la carta si lo desea.

Pues exactamente lo mismo ocurre con las direcciones de Internet. Aunque estas direcciones, las famosas IPs, aparentemente no consten de varios campos (nombre, calle, población, etc.), en realidad esos campos sí que existen, y están codificados dentro del propio número que forma la dirección IP.

Igual que existen las oficinas de correos, también existen una serie de máquinas dedicadas exclusivamente a transportar los "sobres" de Internet de una máquina a otra.
Estas máquinas mediadoras en la comunicación son conocidas habitualmente como routers. Estos routers, al igual que hacen las oficinas de Correos, se dedican a analizar las direcciones Ips para saber dónde tendrán que enviar el "sobre" para que, paso a paso, termine llegando a su destino.



Comparación entre los elementos del correo postal y los elementos de la comunicación de dos máquinas en Internet. Los furgones de correos, los carteros, y las oficinas de correos que forman la red postal, serían los routers que forman la red Internet. Igual que los carteros no abren las cartas, porque les basta sólo con ver el sobre, los routers tampoco ven el interior de nuestros paquetes, si no que miran sólo el equivalente al sobre para saber dónde mandarlos, como veremos más adelante.

3.2. La capa TCP (Transmission Control Protocol = Protocolo de Control de Transmisión) : La necesidad de las conexiones para tener una comunicación fiable

Volvamos ahora al escenario del hospital. En esta ocasión, el doctor PyC recibe un aviso de urgencia a través del servicio de megafonía del hospital.



Pero, ¿qué ocurriría si el doctor PyC no estuviese en ese momento en el hospital? ¿Y si estuviese en esos momentos escuchando música y no oyese el aviso? Cuando se lanza un aviso por megafonía se tiene cierta confianza en que el mensaje llegará a su destinatario, pero en realidad no se puede tener nunca esa certeza. Sueltas el mensaje al vacío, y no tienes forma de saber si al otro lado hay alguien escuchando. Esto es lo que se llama una comunicación no orientada a conexión.

Esta solución en muchos casos no es aceptable, pero en muchos otros sí que puede ser suficiente. Por ejemplo, en el caso de la megafonía del hospital, si el doctor PyC no ha acudido al aviso en un cierto tiempo, probablemente se le volverá a llamar. Existen otros casos de comunicación no orientada a conexión en los que directamente enviamos el mensaje con la esperanza de que llegue pero, si no llega, nos aguantamos y, a otra cosa, mariposa. Un ejemplo es el del envío de postales navideñas. Tú mandas un porrón, y si llegan o no a su destino es algo que en muchos casos nunca sabrás pero, lo que sin duda si que sabes, es que ni por asomo te vas a poner a reenviarlas por si acaso no han llegado.

Como en algunos casos la comunicación no orientada a conexión es suficiente, en las máquinas también es utilizada para algunos casos concretos. Cuando no necesitamos saber si nuestro interlocutor nos está escuchando, no necesitaremos utilizar un protocolo de transporte fiable, como es **TCP**, si no que nos bastará con utilizar un protocolo no orientado a conexión, que también os sonará bastante, y es el **UDP (Protocolo de Datagramas de Usuario)**.

Por tanto, UDP es también un protocolo de transporte e, igual que la mayoría de aplicaciones de comunicaciones utilizan como apoyo TCP/IP, también hay varias aplicaciones que en lugar de eso utilizan como apoyo UDP/IP.

Os remito de nuevo al número 14 de la revista, donde vimos un ejemplo de protocolo apoyado en UDP/IP, que es el protocolo DNS. A lo largo del curso ya veremos en profundidad el protocolo de transporte UDP.



Comentario de Hack x Crack...

Para aquellas mentes inquietas, apuntaremos un detalle.

El protocolo TCP podemos clasificarlo como "pesado" porque, al estar orientado a la conexión, consume muchos más recursos de red, es decir, el proceso para establecer una comunicación de este tipo está formada por muchos pasos.

El protocolo UDP podemos considerarlo como "ligero" porque, al no estar orientado a la conexión, consume pocos recursos de red, es decir, el proceso tiene muy pocos pasos.

El protocolo UDP es muy utilizado, por ejemplo, en la emisión de video por Internet o los programas de intercambio de archivos tipo eMule.

Para que se entienda, en la transmisión de video en tiempo real por Internet (por ejemplo), lo que interesa es que al receptor le llegue la máxima información posible en el menor espacio de tiempo posible, no importa si se pierden unos cuantos frames de película por el camino (es imperceptible), lo importante es que la película no se pare y se pueda ver fluida en tiempo real.

Sería absurdo que si el frame 7 no llega, se parase la película, pidiésemos de nuevo al emisor el frame 7, esperásemos su llegada, la comprobásemos y volviésemos a activar la película. Si pensamos que llegan entre 15 y 30 frames por segundo, bufff, estaríamos parando la película cada dos por tres... es mejor "despreciar" ese frame que no ha llegado y seguir con la "peli" :)

En el caso de los programas de intercambio de archivos tipo P2P (como el eMule, http://www.emule-project.net/), el tema se complica un poquito, pero solo un poquito.

Si estamos descargando un programa llamado "officexp.zip" (de 650MB) desde 7 usuarios a la vez, este nos llega en trocitos pequeños. Lo importante es que nos lleguen cuanto más trocitos mejor y en el menor espacio de tiempo. Pero claro, también es importante que no perdamos ningún trocito (o después el ZIP nos dará errores).

En este caso, podríamos pensar que es mejor utilizar TCP, puesto que nos asegura que llegan todos los trocitos; pero entonces estaríamos sobrecargando la red P2P con centenares de peticiones de comprobación y la descarga sería muy lenta. ¿Cómo resolvemos esto?

Pues trabajamos con UDP y hacemos que sea el programa P2P quien compruebe si faltan trocitos. En caso de faltar algún trozo se reclama y en caso de no faltar no se reclama.

PARALELISMO: Por si alguien no lo ha pillado, retomemos el caso del doctor y hagamos un paralelismo con el "mundo" P2P.

DOCTOR en un sistema TCP en un mundo P2P :)
El doctor (receptor) recibe por megafonía un mensaje (archivo) PERO el tipo del megáfono (emisor) es MUY EXIGENTE y OBLIGA al doctor (receptor) que confirme la correcta recepción de cada palabra (parte del archivo) que recibe.

El mensaje (archivo) a transmitir es: PRESÉNTESE INMEDIATAMENTE EN EL QUIRÓFANO.

1.- El tipo del megáfono (emisor) emite la primera palabra (primera parte del archivo): PRESÉNTESE

2.- El pobre doctor (receptor) va corriendo a un teléfono, llama al tipo del megáfono y le dice que ha recibido correctamente la palabra (trozo de archivo): PRESENTESE

3.- El tipo del megáfono (emisor) emite la segunda palabra (segunda parte del archivo): INMEDIATAMENTE

4.- El pobre doctor (receptor) va corriendo a un teléfono, llama al tipo del megáfono y le dice que ha recibido correctamente la palabra (trozo de archivo): INMEDIATAMENTE

5.- Y así seguiremos hasta que el doctor (receptor) confirma la llegada de la última palabra (trozo de archivo) QUIRÓFANO. En ese momento el doctor (receptor) une todas las palabras (trozos de archivo) y obtiene el mensaje (archivo): PRESÉNTESE INMEDIATAMENTE EN EL QUIRÓFANO.

Como podemos ver, la comunicación es 100% segura, el doctor confirma la llegada de cada palabra (trozo de archivo); pero han invertido mucho tiempo y muchas llamaditas de confirmación.

DOCTOR en un sistema UDP en un mundo P2P :)
El doctor (receptor) recibe por megafonía un mensaje (archivo) PERO el tipo del megáfono (emisor) es MUY DESPREOCUPADO y no le importa si el doctor (receptor) recibe o no el mensaje.

El mensaje (archivo) a transmitir es: PRESÉNTESE INMEDIATAMENTE EN EL QUIRÓFANO.

1.- El tipo del megáfono (emisor) emite la primera palabra (primera parte del archivo): PRESÉNTESE

2.- El doctor (receptor) en teoría recibe la primera palabra (primera parte del archivo): PRESÉNTESE

3.- El tipo del megáfono (emisor) emite la segunda palabra (segunda parte del archivo): INMEDIATAMENTE

4.- El doctor (receptor) en teoría recibe la segunda palabra (segunda parte del archivo): INMEDIATAMENTE

5.- Seguiríamos así hasta que el doctor (receptor) "en teoría" recibiese la última palabra (último trozo del archivo). En ese momento el doctor (receptor) uniría todas las palabras (trozos de archivo) obteniendo el mensaje completo (archivo).

Como podemos ver, la comunicación es mucho más rápida (nos ahorramos las confirmaciones) pero... ¿y si el doctor se ha perdido alguna palabra?... quizás se ha perdido la palabra INMEDIATAMENTE... si el doctor estaba tomándose un café quizá prefiere acabárselo antes de acudir al quirófano (y seguro que su paciente muere desangrado).

En el caso de emisión de video por Internet en tiempo real ya hemos dicho que no nos importaba perder unos cuantos frames, pero en el caso del P2P QUEREMOS TODOS los trocitos de archivo, queremos que el doctor no deje morir a su paciente... ... ¿Cómo lo solucionamos si no queremos utilizar el sistema TCP?

Tenemos un problema: No queremos sobrecargar la red telefónica del hospital confirmando cada palabra que recibimos (TCP) pero queremos asegurarnos de recibir los mensajes completitos. Muy bien, pues en este caso tendremos que dotar a los intervinientes (el Sr. del megáfono y el doctor) de un poquito de inteligencia :)

El Sr. del megáfono (software emisor) y el doctor (software receptor) se reúnen y después de un buen rato discutiendo el problema llegan a una solución. Deciden utilizar el sistema UDP

"pero" cada palabra (trozo de archivo) emitida tendrá estará precedida de un número correlativo (número de control). Vamos a verlo.

DOCTOR en un sistema UDP en un mundo P2P (con control añadido).
El doctor (receptor) recibe por megafonía un mensaje (archivo) con un sistema previamente pactado :)

El mensaje (archivo) a transmitir es: PRESÉNTESE INMEDIATAMENTE EN EL QUIRÓFANO.

Según han pactado, el mensaje a transmitir será: UNOPRESÉNTESE DOSINMEDIATAMENTE TRESEN CUATROEL CINCOQUIRÓFANO.

1.- El tipo del megáfono (emisor) emite la primera palabra (primera parte del archivo): UNOPRESÉNTESE

2.- El doctor (receptor) en teoría recibe la primera palabra (primera parte del archivo): UNOPRESÉNTESE

3.- El tipo del megáfono (emisor) emite la segunda palabra (segunda parte del archivo): DOSINMEDIATAMENTE

4.- El doctor (receptor) en teoría recibe la segunda palabra (segunda parte del archivo): DOSINMEDIATAMENTE

5.- Seguiríamos así hasta que el doctor (receptor) "en teoría" recibiese la última palabra (último trozo del archivo). En ese momento el doctor (receptor/software receptor) comprobaría que tiene en su poder las palabras (trozos de archivo) y que no falta ninguna (se puede comprobar gracias a que tienen números correlativos).

Solo en caso de que faltase alguna palabra (trozo de archivo) el doctor llamaría por teléfono al emisor pidiéndole UNICAMENTE la palabra que le falta.

Como podemos ver, ahora la conexión sigue siendo del tipo UDP (cargamos poco la red); pero gracias a que hemos dotado al Sr. del megáfono y al doctor (software emisor y receptor) de "inteligencia" (software), hemos conseguido además bastante seguridad en la comunicación.

ACABANDO Y PUNTUALIZANDO:

Acabamos de aprender algo importantísimo que ya nunca deberíamos poder olvidar. Una cosa es el tipo de protocolo que estamos utilizando para nuestras conexiones (TCP o UDP e incluso ambos a la vez) y sus consecuencias sobre la red y, OTRA MUY DISTINTA, cómo programamos el software para mejorar el rendimiento de dichas conexiones.

Hemos visto que las carencias de seguridad del protocolo UDP (capa de transporte) han sido "salvadas" gracias a cómo hemos programado el software (nivel de aplicación).

PARA LOS QUISQUILLOSOS:
- Si, pero... ¿y si el doctor (software receptor) no recibe ninguna palabra (trocito de archivo) porque está dormido? Pues entonces dotamos de un poco más de inteligencia al programa para que la primera palabra (trocito de archivo) se haga por TCP (confirmacion aobligatoria) y el resto por UDP. De esta forma no se emitirán mas palabras (trocitos de archivo) por megafonía hasta que el doctor llame al Sr. del megáfono confirmando que la recibido la primera palabra.

- Si, pero... ¿y si el doctor (software receptor) no confirma la recepción de esa primera palabra (trocito de archivo)? Pues hacemos que el Sr. de megafonía (software emisor) envíe un mensaje al teléfono móvil del doctor cada 5 minutos durante 2 horas hasta que conteste.

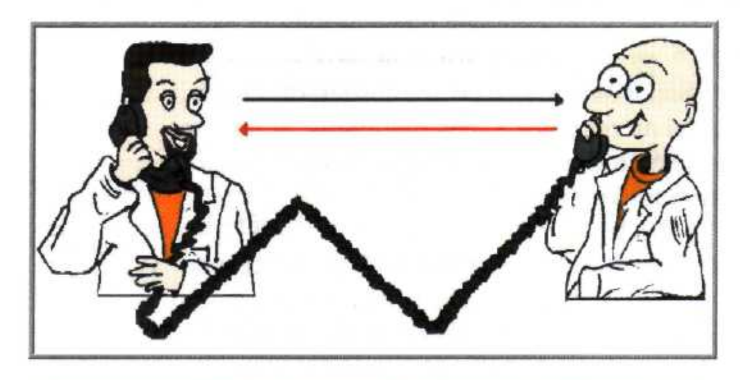
¿Y si a pesar de todo no contesta? Pues llamamos a otro doctor mientras el primero está dormido (en un P2P sería el equivalente a servir el archivo a otro cliente mientras el primero pasa a una lista de espera :)

La intención de esta extensa nota no es otra que ACERCAR ese extraño mundo de las capas OSI a la realidad, a programas que utilizamos diariamente y que no tenemos ni idea de cómo funcionan (por ejemplo la visualización de video en tiempo real y los P2P). Quizás ahora pensemos un poco más en lo que hay detrás de esas cosas que utilizamos mecánicamente sin pensar :)

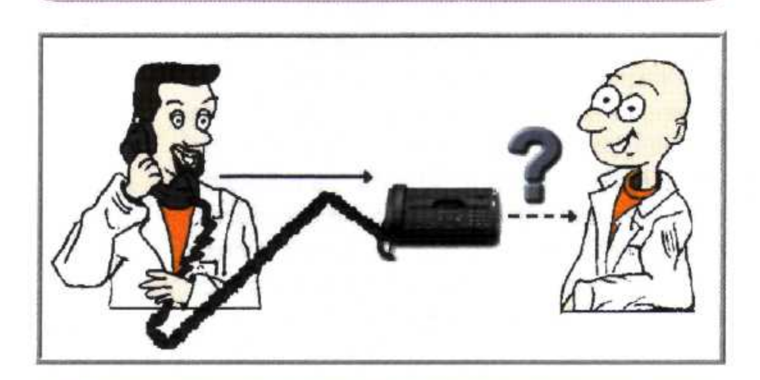
Olvidándonos ya de UDP, vamos a ver entonces qué es TCP, que es el que más nos interesa. A diferencia de las comunicaciones no orientadas a conexión, las orientadas a conexión son aquellas en las cuales hay un diálogo directo con el interlocutor. Es decir, no es ningún monólogo que sueltas con la esperanza de que alguien te escuche, si no que es una conversación entre dos o más

interlocutores, donde todos saben en todo momento si están siendo escuchados por los demás.

Como ejemplo de comunicación orientada a conexión tenemos el teléfono, donde en todo momento sabes si la persona con la que estás hablando está siguiendo el diálogo. Por el contrario, cuando hablas con un contestador automático telefónico, se trata precisamente de una comunicación no orientada a conexión.



Al hablar por teléfono mantenemos una conversación orientada a conexión, donde ambas partes saben que el otro está escuchando.



Al hablar con un contestador telefónico mantenemos una conversación no orientada a conexión, pues no sabemos si la otra parte llegará a escuchar nuestro mensaje.

La mayoría de servicios de comunicación entre máquinas requieren una comunicación orientada a conexión. Por ejemplo, si estas transfiriendo un fichero, normalmente necesitarás saber que éste está siendo recibido, si estás enviando un e-mail necesitarás saber que tu servidor de correo lo está recibiendo (aunque si llega al buzón del destinatario o no es ya un asunto aparte), si estás en un Chat necesitas saber que la persona o personas con las que hablas están conectadas en ese momento, y leyéndote.

Hasta donde la capa IP entiende, sólo existen sobres que circulan desde una dirección de remitente hacia una dirección de destinatario, pero en ningún momento existe un diálogo entre remitentes y destinatarios. Es en la capa TCP donde aparece este nuevo concepto, que engloba los sobres que para IP circulan por separado, en un único flujo de diálogo entre las dos partes.

En el caso de TCP, existen unos "sobres" especiales que lo único que hacen es decir "te estoy escuchando". Cada vez que un remitente envíe un sobre a un destinatario, éste responderá con un nuevo sobre, en el que simplemente dirá "te he escuchado".

Gracias a este mecanismo, se puede saber en todo momento si estamos hablando solos, o estamos en un diálogo.

3.3. La capa TCP: El tamaño de los paquetes.

Pero, ahora que sabemos que el protocolo TCP exige a los destinatarios que constantemente confirmen que han recibido lo que se les enviaba, ¿qué ocurre si en algún momento no se recibe esa confirmación? Pues el remitente, transcurrido un tiempo en el que no haya recibido confirmación, tendrá que reenviar el paquete y esperar de nuevo la confirmación, como en el ejemplo de la megafonía del hospital.

Pero, ¿qué ocurre entonces si el mensaje de megafonía era muy largo? Imaginemos que, en lugar de decir: "Doctor PyC, acuda a la sala de cirugía cardiovascular", el

mensaje dijese: "Doctor PyC, acuda a la sala de cirugía cardiovascular para atender una urgencia de cardiopatía precarótida en un paciente varón de 70 años, diabético, de grupo sanguíneo AB+, y cuyo color preferido es el fucsia". Si el Doctor PyC no respondiese a la primera llamada, habría que repetir toda la parrafada de nuevo.

No tiene sentido soltar parrafadas muy largas si no tienes la certeza de que estás siendo escuchado. Por eso, si lo que tienes que transmitir es muy largo, lo mejor es que lo vayas contando poco a poco, y esperando la confirmación de que cada parte ha sido escuchada.

Cuando hablamos por teléfono, normalmente no soltamos un rollo de varias horas sin parar (aunque los hay que si...), si no que estamos pendientes de que cada cierto tiempo nuestro sufrido interlocutor nos dé las confirmaciones de rigor como "si, si", o "aja", o "que te calles ya". Normalmente, si llevamos dos minutos seguidos hablando y no hemos escuchado un "aja" de nuestro interlocutor, nos mosqueamos bastante, y decimos "oye, ¿sigues ahí?".

En resumen, lo natural a la hora de transmitir mucha información es hacerlo en pequeños trozos, cada uno de los cuales confirmará su recepción por separado.

Lo mismo ocurre en la comunicación entre máquinas. Como TCP se encarga de enviar confirmaciones, es también el que se encarga de partir los paquetes muy grandes en paquetes más pequeños para que estas confirmaciones puedan llegar poco a poco, y no tener que retransmitir todo si no llegase la confirmación.

Esto nos permite, además, adaptarnos a la capacidad de nuestro interlocutor. Por ejemplo, si nos suscribiésemos a una enciclopedia por fascículos, y nos enviasen toda la colección de golpe, probablemente el cartero mandaría al garete a los tíos de Espasa, y les diría que los 20 volúmenes los iba a llevar hasta allí su simpática abuela. Los buzones de correos tienen un tamaño limitado y, si bien cada fascículo por separado cabe perfectamente en el buzón, la colección entera no cabría en ningún buzón. Lo mismo ocurre con las máquinas, que tienen un buzón de recepción de un tamaño limitado, y hemos de ajustarnos a esas limitaciones tecnológicas.

3.4. La capa TCP: Las conexiones simultáneas

Una de las principales funciones de la capa TCP es la de permitir que existan varios diálogos simultáneos entre dos interlocutores. Aquí no recurriré a más metáforas, si no que será más sencillo verlo directamente en nuestro campo de trabajo.

Si, por ejemplo, está PyC en MSN chateando con Scherzo, y a la vez le está enviando un archivo, ¿no estarán manteniendo dos diálogos simultáneos? Por un lado, están chateando, y por otro lado están enviando un archivo.

Suponiendo que un Chat en MSN funcionase mediante una conexión punto a punto (que no es así, como sabréis si habéis leído mi artículo sobre MSN, pero imaginaremos que sí), habría una serie de paquetes cuyo remitente sería PyC y cuyo destinatario sería Scherzo, pero de esos paquetes algunos serían parte del archivo que se está transfiriendo (que, por supuesto, estaría partido en trozos, tal y como vimos en el punto anterior), y otros serían parte de la conversación que mantienen *PyC y Scherzo a través del Chat.*

Para permitir que esto ocurra, el protocolo de transporte, TCP, tiene que tener algún sistema que identifique qué paquetes son del Chat, y qué paquetes son del archivo. Esto lo hace asignando un número a cada diálogo simultáneo y, según el número que haya en cada paquete, sabrá si éste forma parte del archivo, o del Chat.

Pues estos números mágicos de los que estoy hablando no son otros que los archiconocidos **PUERTOS**.

Un puerto es un campo del protocolo TCP que permite identificar el servicio al que va destinado cada paquete en una conexión entre dos máquinas.

Así, cada vez que una máquina reciba un paquete con el número de puerto 25, sabrá que ese paquete es un e-mail, cada vez que reciba un paquete con el número de puerto 21, sabrá que ese paquete es un comando de FTP, cada vez que reciba un paquete con el número de puerto 80 sabrá que es una conexión Web, etc., etc.

4. Ejemplo: Enviando un archivo.

Para recapitular todas las ideas mostradas a lo largo del artículo, termino con un ejemplo bastante completo que muestra paso a paso el envío de un archivo de PyC a Scherzo.

Estad muy atentos a cada paso, porque espero que este ejemplo os ayude a comprender mucho mejor todos los conceptos que necesitareis para seguir el resto del curso. Fijad también vuestra atención en todas las ilustraciones, pues muestran gráficamente toda la secuencia del ejemplo, y además los datos que aparezcan en las propias ilustraciones son también fundamentales.

A lo largo de la serie RAW os he explicado ya varios sistemas de transferencia de archivos (FTP, DCC, MSN,...). En este ejemplo usaremos, por ejemplo, una transferencia por FTP.

Antes de nada, vamos a ver cómo sería el proceso si sólo nos fijásemos en la capa de arriba, es decir, en la capa sobre la que he ido hablando mes tras mes en la serie RAW.

1. **PyC abre su servidor FTP**: pone un puerto en escucha, gracias a una función que da el sistema operativo que permite a cualquier aplicación realizar estas y otras funciones de TCP/IP.
2. **Scherzo abre una conexión con el servidor FTP de PyC**: el modo en que se abre esta conexión lo detallaremos a lo largo del curso, pero no en este artículo. De momento lo que sí que sabemos es que la responsable de abrir y mantener las conexiones es la capa TCP.
3. **Scherzo escoge el archivo que quiere bajar**: comandos CWD, CDUP, LIST,... todo esto ya lo vimos en los artículos sobre FTP de la serie RAW, y ahora no nos interesa mucho.
4. **Scherzo inicia la transferencia del archivo**: comandos PORT o PASV, y RETR o REST. También lo vimos en los artículos de la serie RAW, y tampoco nos interesa ahora.
5. **El archivo se transfiere desde el servidor de PyC hacia el cliente de Scherzo**: Aquí unos enanitos se encargan de llevar el archivo de una máquina a otra, cargando los datos en sacos que llevan a la espalda. Pero... ¡espera! ¡Si esto no es la serie RAW! En la serie RAW no me quedaba más remedio que deciros estas cosas, porque al llegar a este punto no podía daros más detalles, ya que más de una vez os mencioné que explicar lo que ocurre en estos momentos sería suficiente para llenar no sólo un artículo, si no una serie entera. ¡Y al fin ha llegado esa serie! Así que esperad unas cuantas líneas, que enseguida os explico cómo funcionan las cosas realmente. Tampoco quiero chafar la ilusión a nadie, así que si alguien no quiere dejar de creer en los enanitos que transportan paquetes de datos, que no siga leyendo! ;-)
6. **Finaliza la transferencia del archivo:** y a otra cosa, mariposa.

¿Para qué os he mostrado todos estos pasos que conocéis ya perfectamente (sobre todo si habéis seguido la serie RAW)? Pues sencillamente, para que veáis que entre los pasos 5 y 6 ocurren una gran cantidad de cosas que siempre hemos obviado, y que serán las que precisamente detalle en este ejemplo.

Nos olvidaremos, por tanto, del resto de pasos, y nos centraremos únicamente en lo que ocurre

desde que comienza la transferencia del archivo, hasta que ésta finaliza.
Algunos conceptos los simplificaré mucho, y otros incluso los obviaré, esperando así facilitar la comprensión de los conceptos fundamentales, aunque tenga que sacrificar parte de la verdad.

4.1. En el servidor FTP de PyC

Empezamos nuestro viaje por el mundo de los enanitos en el reino que nosotros conocemos, que es el del servidor FTP de PyC.

El servidor lo único que sabe es que tiene un archivo de 4500 KB que tiene que enviar al usuario Scherzo, que previamente hizo un login en el servidor.

El que programó el servidor FTP no tenía ni idea de TCP/IP, así que lo único que sabía era que el no tenía más que dar una orden al sistema operativo, y éste se encargaría de hacer todo el trabajo sucio. Así que el software del servidor simplemente llama a una función mágica del sistema operativo, parecida a esta:

EnviaFichero (fichero.doc, usuario)

Donde fichero.doc es el nombre del archivo que quiere enviar, y usuario es una variable que contiene un número que identifica al usuario Scherzo.

El valor de esa variable usuario no lo asignó el programa de FTP, si no que lo hizo el propio sistema operativo en el momento en que Scherzo se conectó al servidor.

Como es el sistema operativo el que maneja estos números mágicos que identifican a las conexiones (que, como veremos a lo largo del curso, se llaman sockets), el programa de FTP podrá pasarle este dato al sistema, y éste ya sabrá perfectamente lo que hacer con él.

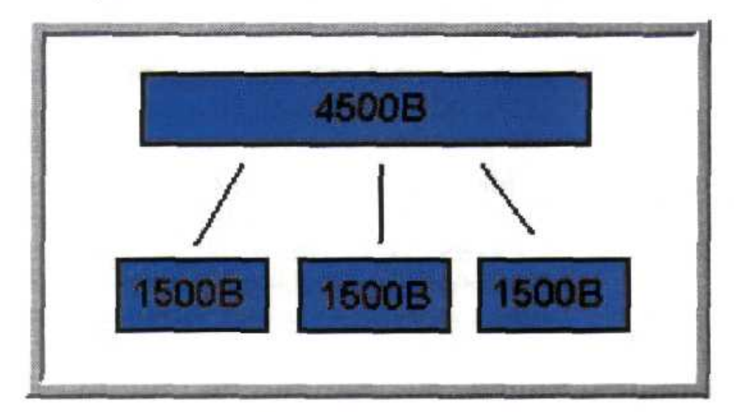
Una vez que el programa FTP llama a esta función mágica del sistema operativo, lo próximo que verá será la respuesta del sistema operativo **diciéndole "ale, ya está transmitido el archivo"**, o bien cualquier error que haya podido ocurrir "**oye, no he podido enviarlo porque el usuario se ha desconectado**", etc.

Como esto es todo lo que ve el programa de FTP, tendremos que descender al oscuro reino del sistema operativo para comprender mejor qué es lo que ocurre desde que se llama a esa función hasta que el sistema operativo avisa de que el envío ha finalizado.

4.2. En el Sistema Operativo de PyC: La capa TCP.

Lo primero que hace la capa TCP nada más ver el archivo de 4500KB es decir: "buf, este chorizo es demasiado largo". ¿Qué criterio utiliza TCP para decidir si algo es demasiado largo? Es algo que veremos a lo largo del curso, pero de momento nos basta con saber que el tamaño máximo de un paquete es algo que determinó previamente la capa TCP según las limitaciones de la red.

Supongamos que ha determinado que el tamaño máximo de paquete son 1500KB. Tendrá que coger el archivo y partirlo en tres trozos, cada uno de 1500KB.



A cada trozo le asigna un número que determina la secuencia en la que han de unirse de nuevo los trozos. A este número se le llama precisamente número de secuencia.

Una peculiaridad del número de secuencia es que no se numera según el número de trozo,

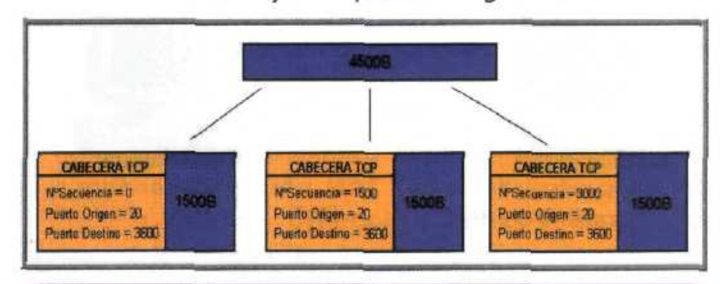
si no según el número de byte del archivo en el que empieza ese trozo. Así, el primer trozo tendrá un número de secuencia 0, el segundo un número de secuencia 1500, y el tercero un número de secuencia 3000.

Pero, ¿es éste el único dato que ha de asignar la capa TCP a cada trozo? Pues me temo que no, ya que sabemos bien que las responsabilidades de esta capa van más allá de simplemente partir los paquetes en bloques. Hablamos también de los números de puerto, así que tendrá que añadir a cada trozo los dos puertos implicados en la conexión. ¿Adivináis cuáles son estos puertos?

Je, je... era una pregunta con trampa. Si habéis pensado que uno de los puertos era el 21, puerto asignado al servicio FTP, os habéis equivocado, aunque he de reconocer que lo he preguntado a mala leche. 0:-)

Si repasáis mejor el artículo de la serie RAW sobre FTP veréis que el puerto 21 sólo se utiliza para enviar comandos al servidor, pero no para la transferencia de archivos. Para ésta se utiliza otro puerto que es acordado previamente mediante el comando **PORT** o el comando **PASV**. Para los que no sepáis de qué hablo porque no conocéis el protocolo FTP, olvidaos de todo esto y quedaos simplemente con la idea de que tanto el puerto de PyC (origen) como el puerto de Scherzo (destino) son números que han acordado previamente, por ejemplo, el 20 y el 3600.

Por tanto, si añadimos a cada trozo los tres numerajos que hemos dicho (numero de secuencia, puerto de origen, y puerto de destino) *nos queda el siguiente escenario:*



Los tres paquetes que forman el archivo, con sus correspondientes cabeceras TCP. Fijaos bien en los campos que forman la cabecera.

La capa TCP ya ha terminado su trabajo inicial, y de momento se puede relajar un poco mandando los bloques a la capa IP, que sabrá qué hacer con ellos a continuación. Pero, a diferencia del programa de FTP, la capa TCP no se puede dormir en los laureles esperando a que la transferencia del archivo termine, si no que tendrá que seguir trabajando más adelante, tal y como iremos viendo.

4.3. En el sistema operativo de PyC: La capa IP.

En cuanto TCP llama a la capa IP, y le pasa los 3 bloques, ésta se pone en marcha. Su labor principal consiste en añadir a cada bloque las direcciones IP de PyC y Scherzo para que, una vez que los bloques estén flotando por el ciberespacio, todos aquellos mediadores por los que pasen sepan dónde llevarlos.

La dirección IP de PyC la conoce, por supuesto, el propio sistema operativo de PyC (bien porque la introdujimos nosotros manualmente al configurar nuestra LAN, bien porque el sistema la asignó automáticamente mediante DHCP, o bien porque se nos asignó una IP dinámica al conectar con nuestro ISP).

La dirección IP de Scherzo la puede obtener el sistema a partir de la variable usuario ya que, cuando Scherzo conectó con el programa FTP, el sistema operativo automáticamente asoció la IP de Scherzo a esa variable.

¿Tiene que añadir algo más la capa IP? ¡Pues sí! Vamos a ver. ¿Para qué servía en la capa TCP meter los números de puerto en cada bloque? Si no metiésemos los números de puerto, cuando el bloque llegase al destinatario (Scherzo) éste no sabría qué hacer con él. Scherzo probablemente tendría abiertas en ese momento varias aplicaciones de comunicaciones, y cada paquete que le llegase tendría que indicar de algún modo a cuál de todas esas aplicaciones iba dirigido.

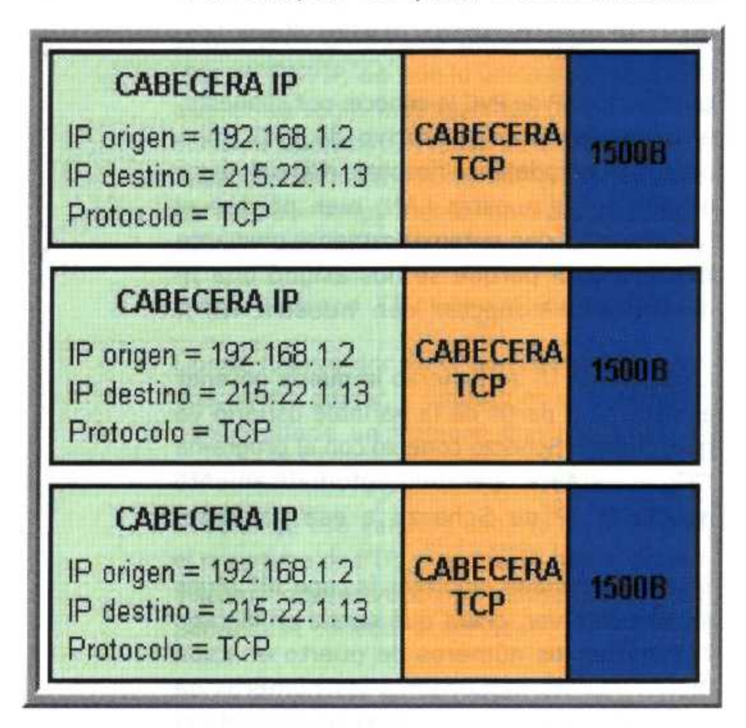
Pues algo parecido ocurre con la capa IP. Como ya vimos, no sólo existe TCP como protocolo

de transporte, si no también otros como UDP. Además, existen otra serie de protocolos que funcionan sobre IP, como ICMP, pero de eso no vamos a hablar ahora.

Lo que ahora nos interesa es que de alguna manera el bloque tiene que decir de alguna forma que es un bloque TCP, y no UDP. Para ello, la capa IP asigna mete un numerajo en cada bloque que identifica el protocolo de transporte desde el que le llegó ese bloque.

Como, en este caso, los bloques le llegaron a IP desde la capa TCP, la capa IP meterá en cada bloque un número que dirá que ése es un bloque TCP.

Por tanto, así nos queda el escenario ahora:



Los tres paquetes que forman el archivo, con sus cabeceras TCP e IP. Como podéis observar, la cabecera IP es igual para los tres paquetes.

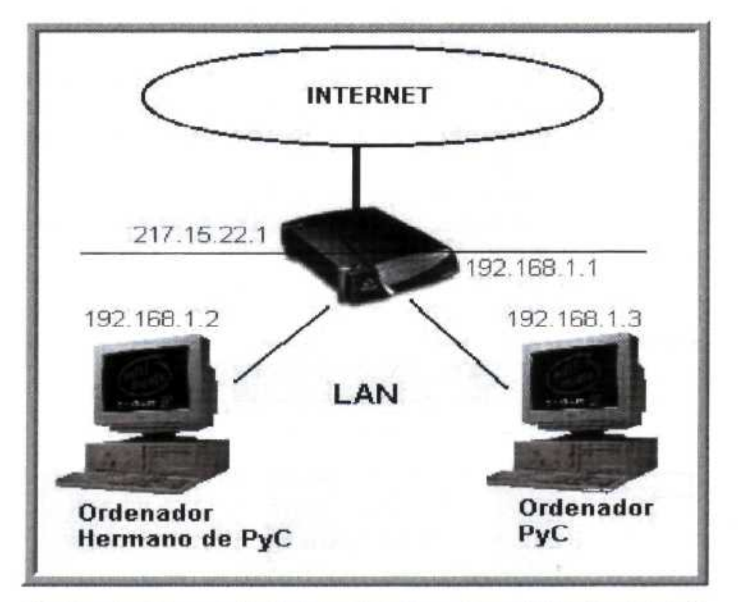
Una vez que los paquetes ya están listos, se los manda a la capa de abajo, la capa de enlace, de la que hemos hablado poco hasta el momento.

4.4. En el sistema operativo de PyC: la capa de enlace.

Hasta el momento hemos hablado poco de esta capa para evitar liaros más. Si bien las capas TCP e IP son iguales para todos los sistemas, la capa de enlace puede variar totalmente de un sistema a otro. Es aquí donde se encuentran las diferencias entre una conexión por módem, una conexión ADSL, y cualquier otro tipo de conexión.

Es decir, la capa de enlace depende de la tecnología utilizada para crear el enlace entre nuestra máquina e Internet.

Vamos a suponer que el enlace que tiene PyC es el más común hoy día, es decir, un enlace Ethernet. Éste es el que usamos si tenemos una tarjeta de red (bien con ADSL, o con cable, o con otras tecnologías).
Supongamos que PyC tiene en su casa una configuración muy habitual hoy día, que es la de una conexión a Internet a través de un router ADSL, con una pequeña LAN (red local) en casa compuesta por dos ordenadores: el suyo, y el de su hermano.



Esquema de la LAN de PyC, con las IPs de cada elemento. El router ADSL tiene 2 IPs: 192.168.1.1 para la LAN, y 217.15.22.1 de cara a Internet.

Como vemos en la ilustración, la única máquina que se comunica directamente con Internet es el router ADSL, y es éste el que tiene que encargarse de llevar todos los paquetes de PyC y de su hermano a Internet. Para ello, ambos ordenadores están conectados al router mediante un cable de red (si el router sólo tiene un conector RJ-45 tendría que haber en medio un switch, pero esa cuestión la obviamos por salirse del tema).

El problema aquí es que la comunicación en Ethernet es de tipo **broadcast**.

Esto significa que lo que circula por el cable llega a todas las máquinas que estén conectadas al mismo, y hay que idear alguna forma de hacer que sólo atienda a los datos la máquina interesada.

Este es precisamente el motivo por el que funciona un sniffer en una red local. Todos los datos de la red local circulan por el cable al que se conecta tu tarjeta de red, y sólo hay que "engañar" a la tarjeta de red para que te muestre todos los datos que hay en el cable, y no sólo los que van dirigidos a ella.

Pero lo que nos interesa aquí es conocer el mecanismo utilizado para distinguir unas máquinas de otras en una LAN.

Como podéis imaginar, esto se consigue asignando un número diferente a cada tarjeta de red de las máquinas conectadas a la LAN. Este número es único en el mundo para cada tarjeta de red.

Los fabricantes de dispositivos Ethernet tienen un acuerdo para que nunca se fabriquen dos tarjetas de red con el mismo número. Este *número mágico es precisamente la famosa* **dirección MAC (comúnmente conocida como dirección física)** de la que probablemente habréis oído hablar.

Al haber una MAC diferente por cada dispositivo Ethernet del mundo, las direcciones MAC tienen que ser lo suficientemente grandes como para que nunca tengan que repetirse. Estas direcciones, más largas que las direcciones IP, constan de 48 bits, por lo que teóricamente permiten identificar casi 300 Billones de dispositivos Ethernet diferentes. Entonces, ¿qué datos añadirá el nivel Ethernet a cada bloque que queremos transmitir? Pues, al igual que la capa IP, añadirá una dirección MAC de origen, y una dirección MAC de destino. Y, también igual que en la capa IP, tendrá que añadir un dato más, que es un identificador de la capa superior que le pasó los bloques, es decir, en este caso la capa IP.

CABECERA ETHERNET			
MAC origen = 00045A44B899 MAC destino = 009027F69AD4 Protocolo = IP	CABECERA IP	CABECERA TCP	1500B

CABECERA ETHERNET			
MAC origen = 00045A44B899 MAC destino = 009027F69AD4 Protocolo = IP	CABECERA IP	CABECERA TCP	1500B

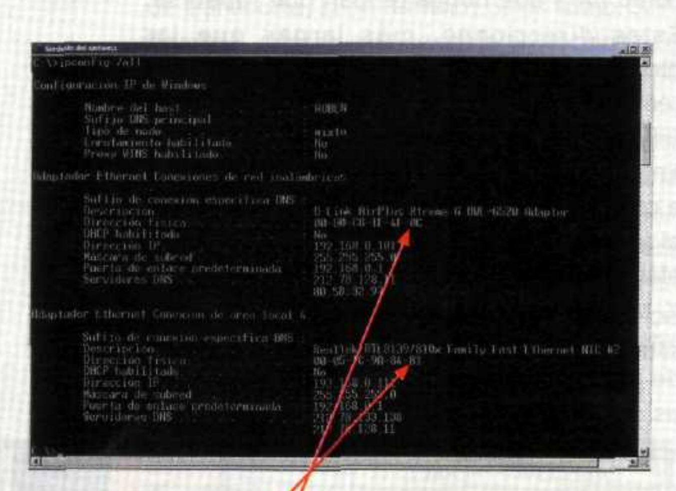
CABECERA ETHERNET			
MAC origen = 00045A44B899 MAC destino = 009027F69AD4 Protocolo = IP	CABECERA IP	CABECERA TCP	1500B

Los tres paquetes que forman el archivo, con sus cabeceras TCP, IP, y Ethernet. En ésta la MAC de origen será la de PyC, y la MAC de destino la del router ADSL de PyC, que será el próximo punto con el que habrá que enlazar la comunicación. El ordenador de PyC conoce la dirección MAC del router gracias al protocolo ARP, pero eso se sale del tema del artículo.

¡No sabes...

¿No sabes la MAC de tu tarjeta de Red? ¿de verdad?... bueno, bueno... tendrías que haber leído los anteriores números de esta revista :)

Abre una ventana de comandos (Menú inicio --> Todos los Programas --> Accesorios --> Símbolo del sistema). Escribe IPCONFIG /ALL. Pulsa enter y ZAS!!! Ahí tienes la MAC de tu/s tarjetas Ethernet (Tarjetas de Red).



En este caso podemos ver 2 tarjetas de red y cada una tiene una MAC (dirección física). La tarjeta D-Link AirPlus tiene la MAC 00-80-C8-1E-4F-0C y la Realtek tiene la MAC 00-05-1C-9A-84-B1

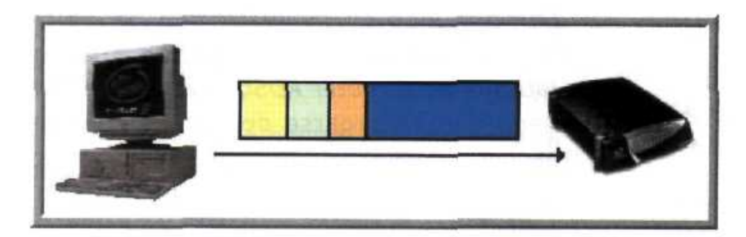
Como apunte interesante, aclarar que aunque la MAC es única en el mundo, existen Tarjetas de Red especiales y otros dispositivos (como algunos routers) que permiten "falsear" la MAC (poner la MAC que tu quieras ;). La mayoría de técnicos de Red tienen entre sus "herramientas de trabajo" una de estas tarjetas para comprobar el buen funcionamiento de los Sistema de Red.

4.5. En la tarjeta de red de PyC: La capa física

Una vez que el nivel de enlace ya ha hecho lo que tenía que hacer con los bloques, no tiene más que acceder ya directamente al hardware, en este caso la tarjeta de red, y los datos salen al fin de la máquina de PyC.

Por supuesto, existen muchas tecnologías diferentes en la capa física: RJ-45 (la que casi todos tenemos), coaxial, inalámbricas (muy de moda últimamente), etc., etc.

Una vez lanzados los paquetes a través del cable que une la máquina de PyC con el router ADSL estos, por supuesto, llegarán hasta el router.



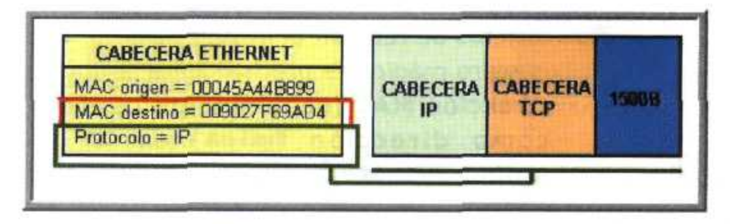
El primer paquete al fin sale del ordenador de PyC, hacia el router ADSL.

4.6. En el router ADSL de PyC: Las capas física y de enlace

Una vez que los datos llegan al router, éste detectará mediante hardware la presencia de datos en el cable, y enviará los bloques recibidos a su capa de nivel de enlace. Aquí es importante tener en cuenta que el nivel de enlace del router debe ser también Ethernet. De no ser así, la comunicación entre PyC y su router sería imposible.

Por tanto, una vez que los datos llegan a la capa Ethernet del router, éste analiza la cabecera Ethernet del paquete.

Al analizar la cabecera, descubre que la dirección MAC de destino era la suya, por lo que decide continuar el procesamiento del paquete, ya que le corresponde a él hacerlo. Para poder continuar el procesamiento debe saber a qué capa pasarle la pelota. Para ello analiza la cabecera Ethernet del paquete y descubre la capa de encima tiene que ser la capa IP. Así, el router elimina la cabecera Ethernet del paquete (ya que la capa IP es incapaz de comprenderla), y pasa el resto del paquete a la capa IP.



El router analiza la cabecera Ethernet, comprobando que la MAC de destino es la suya, y pasa el resto del paquete (sin la cabecera Ethernet) a la capa IP.

4.7. En el router ADSL de PyC: La capa IP

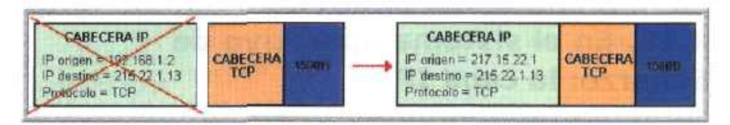
Una vez que la capa IP recibe el paquete, analiza su cabecera. Ahí encuentra la dirección IP de destino, que es la de Scherzo. El router tiene configurada internamente una tabla parecida a esta:

IP Routing Table			
Type	**Destination**	**Netmask**	**Gateway**
Network	192.168.1.0	255.255.255.0	192.168.1.1
Network	217.15.22.64	255.255.255.0	217.15.22.1

Lo que vemos en esta tabla es que el router ADSL enviará al router de su ISP cualquier paquete que tenga como IP de destino una que no pertenezca a la LAN de PyC. Como la dirección IP de Scherzo no es una de las de la LAN de PyC, el paquete será enviado al router del ISP de PyC.

Una vez encontrado el siguiente mediador al que tiene que enviar el paquete, el trabajo del router ADSL termina aquí. No le interesa para nada lo que haya dentro del paquete, es decir, la cabecera TCP, y los datos del archivo que estamos enviando. Eso ya será problema del PC de Scherzo cuando reciba el paquete.

Lo único que tiene que hacer ahora el router ADSL es devolver el paquete a la capa de enlace para que ésta sepa qué hacer con él.



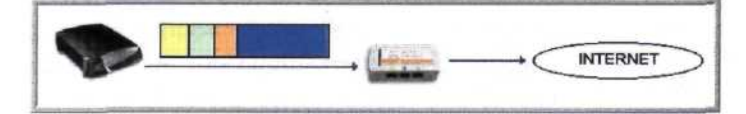
El router modifica la cabecera IP, sustituyendo la IP de LAN de PyC por la IP pública para que pueda salir ya al resto de la red Internet. Ya explicaremos esto mejor a lo largo del curso.

4.8. En el router adsl de PyC: De nuevo en la capa de enlace.

Pero... ¡Sorpresa! Con quien ha contactado la capa IP del router no es con la capa Ethernet, si no con otra capa de enlace diferente!!

Esto es debido a que la conexión entre el router ADSL y el router del ISP no es mediante un enlace Ethernet, si no mediante una tecnología de enlace ADSL.

No vamos a detallar aquí más capas de enlace, o buen barullo montaríamos, así que nos imaginaremos que esta capa añadirá sus propias direcciones y su identificador de la capa superior, y enviará el paquete a la capa física, que esta vez no será el cable Ethernet, si no el cable telefónico que va hacia el splitter.



El paquete ya puede salir a Internet, una vez reconstruidas las cabeceras IP y de enlace. La cabecera TCP y los datos del archivo (zona azul del paquete) se han mantenido intactas.

4.9. En el router del ISP de PyC

Una vez que el paquete llega al router del ISP, éste repetirá el proceso de recoger el paquete, analizar desde la capa de enlace si va dirigido a él, pasarlo a la capa IP, mirar su tabla de encaminamiento interna, y decidir a qué mediador enviar el paquete a continuación.

Imaginemos que ésta es una parte de la tabla de encaminamiento de este router:

IP Routing Table			
Type	**Destination**	**Netmask**	**Gateway**
Network	215.0.0.0	255.0.0.0	215.12.133.2

En función de esta tabla, el router decidirá enviar el paquete a otro router, cuya IP es 215.12.133.2. Así, construirá una nueva cabecera de enlace, y reenviará el paquete por el cable físico adecuado.

4.10. Flotando en el ciberespacio

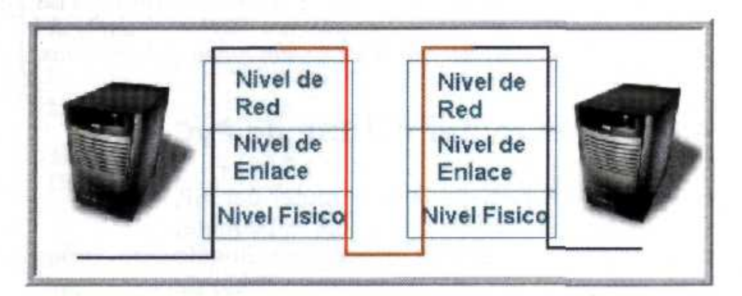
Este proceso de saltar de un router a otro buscando un camino que llegue hasta Scherzo puede constar de muchos pasos. Si queréis hacer la prueba vosotros mismos, tenéis un comando en vuestro sistema que os permite ver por qué routers van pasando vuestros paquetes al enviarlos a un destino concreto.

Ya os explicaré cómo funciona internamente este comando, pero de momento haced la prueba en una ventana MS-DOS (Ventana de Comandos), o una consola Linux:

tracert www.google.com

Cada línea es uno de los routers que forman el camino entre vosotros y la Web de Google.

En cada uno de estos routers se repetirá todo el proceso de analizar el paquete hasta la capa IP, e ir relanzándolo por los distintos cables que forman la auténtica telaraña de Internet.



Cada router por el que pase el paquete analizará sus cabeceras de enlace e IP, para ir retransmitiendo el paquete a cada punto que forma el camino entre el origen y el destino.

Cuando, a lo largo del curso, hable sobre la capa TCP, veremos que los paquetes no siguen siempre el mismo camino, e incluso pueden llegar desordenados a su destino, pero de momento no vamos a entrar en tanto detalle, que me falta ya espacio. ;-)

4.11. En el router ADSL de Scherzo

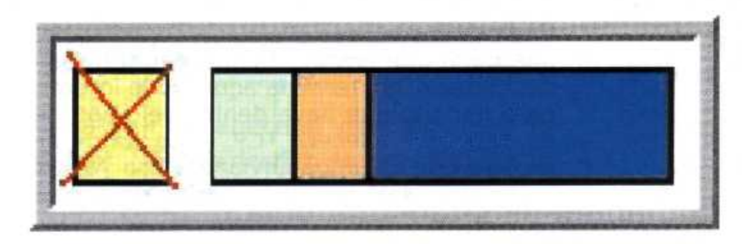
¡Al fin el paquete llegó hasta el último router del camino! Este, por supuesto, es el router ADSL de Scherzo. Éste analizará el paquete, y verá que en la capa IP tiene como dirección IP de destino la IP de Scherzo, así que ahora sólo le falta saber a cuál de los PCs de la LAN de Scherzo enviarlo.

Todos los PCs de la LAN de Scherzo comparten una misma IP de cara a Internet, que es la IP del router, y éste los diferencia sólo por su IP de LAN y por su dirección MAC. El cómo sabe el router a qué máquina enviar el paquete se sale ya del tema que puedo abarcar por hoy, así que nos creeremos que sabe dirigir el paquete a su destino dentro de la LAN.

Para ello, construye una nueva cabecera Ethernet (capa de enlace) que tiene como dirección MAC de destino la dirección MAC de Scherzo.

4.12. En el PC de Scherzo: la capa Ethernet.

Una vez que el paquete llega a Scherzo, su tarjeta de red empieza analizando la capa de enlace (Ethernet) y, al ver que la dirección MAC de destino es la suya, sabe que el paquete es para él. A continuación, lee el campo de la cabecera Ethernet que le dice que la siguiente capa que tiene que analizar el paquete es la capa IP, y le pasa la pelota a esta capa del sistema operativo.

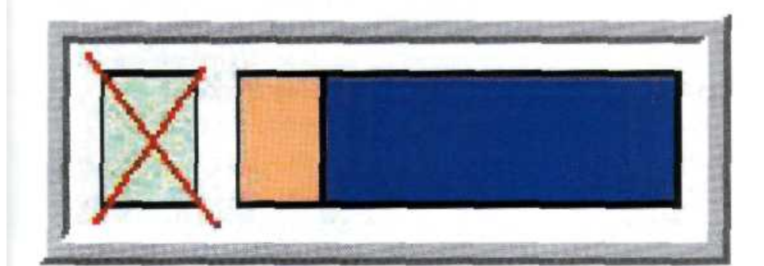


Una vez analizada la cabecera Ethernet, se elimina, y se pasa el resto del paquete a la capa IP.

4.13. En el sistema operativo de Scherzo: la capa IP

La capa IP de Scherzo analizará ahora no sólo la IP de destino (que es la de Scherzo), si no también la de origen (que es la de PyC), ya que tendrá que enviar sus respuestas a esa dirección. Una vez que se queda con estos dos datos, manda el paquete a la capa superior que, según la cabecera IP, es la capa TCP.

En todos los saltos que ha ido dando el paquete de un router a otro, ninguno ha analizado su cabecera TCP, ya que esto es sólo responsabilidad del destinatario final (Scherzo).



Una vez analizada la cabecera IP, se elimina, y se pasa el resto del paquete a la capa TCP.

4.14. En el sistema operativo de Scherzo: la capa TCP

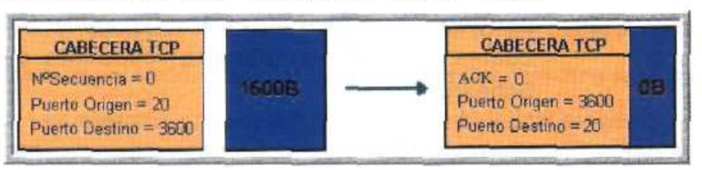
La capa TCP de Scherzo cogerá el paquete, y verá que no es un paquete completo, si no sólo un trozo (recordemos que el archivo se partió en 3 trozos en la capa TCP de PyC).

La capa TCP de Scherzo tiene ahora dos responsabilidades: responder a PyC diciéndole que ha recibido el primer trozo, y mandar el paquete a la capa de arriba, es decir, a la aplicación cliente de FTP, que será la que sepa qué hacer con los datos contenidos en el paquete.

Con respecto al segundo punto, poco hay que decir. Simplemente, se eliminará la cabecera TCP y lo que quedará será un paquete de datos limpio y sin ninguna cabecera, que es lo que comprende la aplicación de PyC. La capa TCP esperará a tener el resto de trozos del archivo para poder reconstruirlo en el orden adecuado gracias a los números de secuencia.

Con respecto al primer punto, la capa TCP de Scherzo tiene que avisar a PyC de que ha recibido el primer trozo, así que comienza la construcción de un paquete totalmente nuevo.

Este paquete simplemente tendrá un aviso del tipo "oye, que te estoy escuchando, y he recibido ese paquete". Para ello, tiene una cabecera TCP especial que incluye, además del aviso de que está escuchando, un dato que es el último byte recibido del archivo. Como hemos recibido de momento sólo el primero de los tres trozos, habremos recibido hasta el byte 1500 del archivo. Por tanto, construimos una cabecera como esta:



El sistema de Scherzo construye un nuevo paquete para indicar a PyC que recibió el suyo. La cabecera TCP de este nuevo paquete constará de un campo ACK con el mismo valor que el número de secuencia del paquete al que quiere responder, e intercambiará los puertos de origen y de destino. El contenido del paquete (zona azul) estará vacío, ya que lo único importante aquí son las cabeceras.

4.15. En el sistema operativo de Scherzo: de vuelta en la capa IP

Esta cabecera la pasaremos a la capa IP, que conoce la IP de PyC, por lo que construye una cabecera IP adecuada para ese paquete:

CABECERA IP	CABECERA TCP	
IP origen = 215.22.1.13 IP destino = 192.168.1.2 Protocolo = TCP	ACK = 0 Puerto Origen = 3600 Puerto Destino = 20	0B

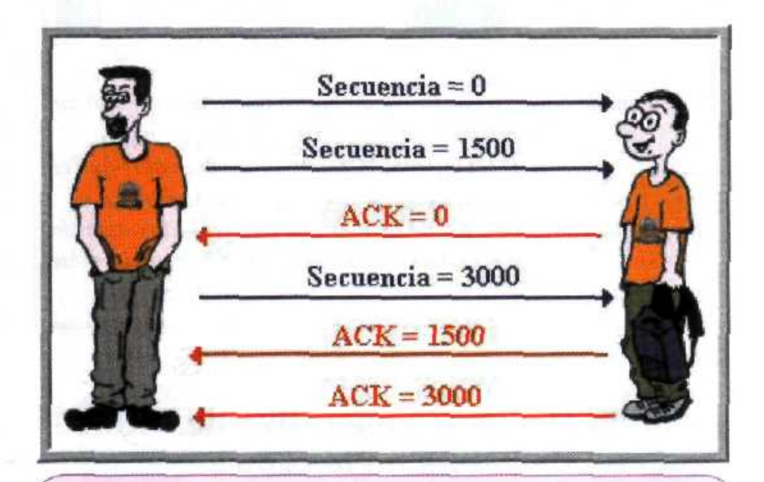
En la cabecera IP del nuevo paquete también se intercambian las IPs de origen y de destino.

4.16. ¡Vamos pa atrás!

Y así continúa el proceso, llevando el nuevo paquete de vuelta por el camino inverso al que siguió el paquete de PyC, hasta que el nuevo paquete llegue a PyC, éste analice su cabecera TCP, descubra que el primer paquete que envió llegó con éxito a su destino, y se despreocupe así ya al fin del paquete de marras.

Normalmente, no esperará a que Scherzo le vaya avisando de que recibe cada paquete, si no que enviará varios de golpe y, sólo si no recibe alguna de las respuestas, entonces reenviará aquel o aquellos de los que no ha recibido respuesta.

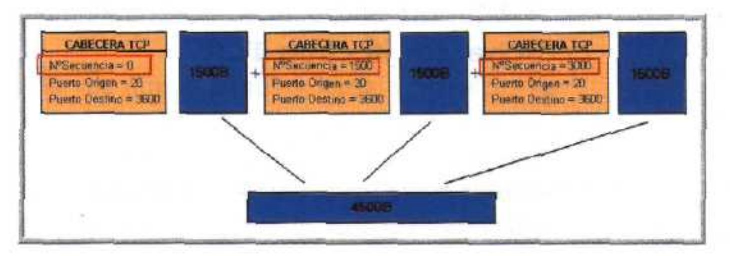
Por tanto, es probable que para cuando PyC haya recibido la primera respuesta de Scherzo, ya haya enviado también los otros dos paquetes que forman el archivo, ante los cuales también esperará la respuesta de Scherzo.



PyC envía los tres paquetes y espera un tiempo razonable a que le llegue la respuesta (ACK) de Scherzo diciendo que ha recibido cada uno de los paquetes

4.17. ¡Conseguido!

Una vez que Scherzo tiene ya los tres trozos del archivo, ya no tiene más que unirlos en el orden adecuado, el cual conoce gracias al número de secuencia de cada trozo, reconstruyendo así el archivo original, y mandándolo a la aplicación de FTP de Scherzo sin que ésta tenga ni idea de cómo ha llegado hasta allí ese archivo, confiando en que unos enanitos mágicos se habrán encargado del trabajo sucio. ;-)



Gracias al número de secuencia de cada paquete se puede reconstruir el archivo original, uniendo cada fragmento en el punto (posición en bytes) indicado por el número de secuencia.

Autor: PyC (Lco)

Ilustraciones: Adhara (Lco)

Curso de PHP (4ª Entrega)
Aprende a manejar cadenas de texto en PHP

En PHP es muy importante que sepamos tratar las cadenas de texto. Comprobaremos con ejercicios practicos la utilidad de conocer estos métodos. Haremos una Web tipo GOOGLE al estilo Hack x Crack :)

Ya va siendo hora de que aprendamos a manejar las cadenas de texto, PHP ofrece funciones muy potentes para la manipulación cadenas. Se van a enumerar la gran mayoría de las funciones, son muchas por lo que te puede resultar un poco aburrido, pero es importante que conozcas que existen, ya que te ahorrarás trabajo.

Eso haremos en este capítulo y además finalizaremos con un ejemplo práctico, ¿cuál será la práctica?, pues vamos a crear una versión de Google al estilo de hackxcrack, donde nosotros pondremos la primera piedra y después esperamos que lo completes.

Mayúsculas y minúsculas:

strtoupper

Convierte un *string* a mayúsculas y devuelve un *string*. La sintaxis es la siguiente:

string = strtoupper(string);

```
<?php
$cadena="hackxcrack es tu revista";
print strtoupper($cadena); // Da como salida "HACKXCRACK ES TU
REVISTA"
?>
```

strtolower

Convierte un *string* a minúsculas y devuelve un *string*.
La sintaxis es la siguiente:

string = strtolower(string);

```
<?php
$cadena="HACKXCRACK ES TU REVISTA";
print strtolower($cadena); // Da como salida "hackxcrack es tu revista"
?>
```

ucfirst

Pasa a mayúscula el primer carácter de un *string* y devuelve un *string*.
La sintaxis es la siguiente:

string = ucfirst(string);

```
<?php
$cadena="hackxcrack es tu revista";
print ucfirst($cadena); // Da como salida "Hackxcrack es tu revista"
?>
```

ucwords

Pasa a mayúsculas el primer carácter de cada palabra de un *string* (separadas por blancos, tabulaciones y saltos de línea)
La sintaxis es la siguiente:

string = ucwords(string);

```
<?php
$cadena="hackxcrack es tu revista";
print ucwords($cadena); // Da como salida "Hackxcrack Es Tu Revista"
?>
```

Trimming:

chop

Elimina blancos y saltos de línea a la derecha de un *string* dado.
La sintaxis es la siguiente:

string = chop(string);

```
<?php
$cadena="hackxcrack es tu revista     "; // ponemos 5 espacios a la
//derecha
print chop($cadena); // Da como salida "hackxcrack es tu revista", ha
//eliminado los espacio de la derecha
?>
```

ltrim

Elimina blancos y saltos de línea a la izquierda de un *string*.
La sintaxis es la siguiente:

string = ltrim(string);

```
<?php
$cadena="     hackxcrack es tu revista"; // ponemos 5 espacios a la izquierda
print ltrim($cadena); // Da como salida "hackxcrack es tu revista", ha
//eliminado los espacio de la izquierda
?>
```

trim

Elimina blancos y saltos de línea a derecha e izquierda de un *string*.
La sintaxis es la siguiente:

string = trim(string);

```
<?php
$cadena="     hackxcrack es tu revista     "; // ponemos 5 espacios a la
//izquierda y derecha
print trim($cadena); // Da como salida "hackxcrack es tu revista", ha eliminado
//los espacio de la izquierda y derecha
?>
```

Comparaciones:

strpos

Devuelve la posición de la primera ocurrencia de *string* 2 dentro de string1.

La sintaxis es la siguiente:

int = strpos(string1,string2);

```
<?php
$cadena="hackxcrack es tu revista";
print strpos($cadena,"x");
?>
```

Da como resultado 5, que es la posición en donde se encuentra la primera ocurrencia de la letra "x".

strspn

Devuelve la longitud en caracteres de string1 contando desde el principio hasta que aparece un caracter en string1 que no esta en string2.

La sintaxis es la siguiente:

int = strspn(string1,string2);

```
<?php
$cadena="hackxcrack es tu revista";
print strspn($cadena,"hkca");
?>
```

El ejemplo anterior da como resultado el valor 4, corresponde a la letra x de la cadena string, cuya letra no se encuentra en la cadena hkca del string2. Recuerda que el valor 0 de la cadena corresponde a la primera posición de la cadena.

strcspn

Devuelve la longitud de string1 desde el principio hasta que aparece un caracter que pertenece a string2.

La sintaxis es la siguiente:

int = strcspn(string1,string2);

```
<?php
$cadena="hackxcrack es tu revista";
print strcspn($cadena,"r"); // da como resultado el valor 6, que es la posición
//en donde se encuentra r
?>
```

strcmp

Compara dos strings y devuelve 1, 0 o -1 según sea mayor el primero, iguales o el segundo.

La sintaxis es la siguiente:

int = strcmp(string1,string2);

```
<?php
$variable1="2";
$variable2="9";
print strcmp($variable1,$variable2);
?>
```

Da como resultado -1, ya que el valor de la variable $variable2 es mayor que $variable1.

strcasecmp

Idem anterior pero case-insesitive (no distingue mayúsculas y minúsculas)

La sintaxis es la siguiente:

int = strcasecmp(string1,string2);

strstr

Devuelve el número de caracteres de string1 desde la primera ocurrencia de string2 hasta el final.

La sintaxis es la siguiente:

int = strstr(string1,string2);

```
<?php
$cadena1="esto es un ejemplo de strstr";
$cadena2="ejemplo";
print strstr($cadena1,$cadena2);
?>
```

Da como resultado "ejemplo de strstr".

stristr

Idem anterior pero case-insensitive (no distingue mayúsculas de minúsculas)

La sintaxis es la siguiente:

int = stristr(string1,string2);

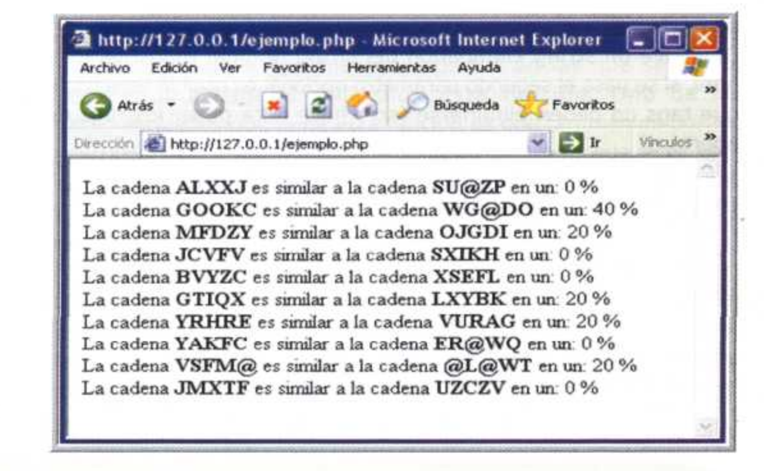
similar_text

Analiza la semejanza entre dos strings, devuelve la cantidad de caracteres iguales en los dos strings, si se pasa como tercer parámetro una referencia a una variable devuelve en la misma el porcentaje de similitud entre ambos strings de acuerdo al algoritmo de Oliver (1993).

La sintaxis es la siguiente:

int = similar_text(string1,string2,referencia);

```
<?php
for ($bucle=0; $bucle<10; $bucle++) {
$cadena1=chr(rand(64,90)).chr(rand(64,90)).chr(rand(64,90)).chr(rand(64,90)).chr(rand(64,90));
$cadena2=chr(rand(64,90)).chr(rand(64,90)).chr(rand(64,90)).chr(rand(64,90)).chr(rand(64,90));
similar_text($cadena1,$cadena2,$valor);
print "La cadena <b>$cadena1</b> es similar a la cadena <b>$cadena2</b> en un: $valor %<br>";
}
?>
```



Funciones de Separación (Parsing):

explode

Devuelve un vector donde cada elemento del vector es un substring del string pasado particionado de acuerdo a un cierto caracter separador.
La sintaxis es la siguiente:

array=explode(separator,string);

```
<?
$st="hola,mundo"
$vec=explode(",",$st); /$vec=("hola","mundo");
?>
```

implode

Genera un string concatenando todos los elementos del vector pasado e intercalando separador entre ellos.
La sintaxis es la siguiente:

string = implode(separator,array);

chunk_split

La sintaxis es la siguiente:

string = chunk_split(string,n,end);

end es opcional y por defecto es "\r\n", devuelve un string en donde cada "n" caracteres del string original se intercala el separador "end".

```
<?
$st="hola mundo";
$st2=chunk_split($st,2,","); //$st2="ho,la, m,un,do";
?>
```

count_chars

Devuelve un vector de 256 posiciones donde cada posición del vector indica la cantidad de veces que el carácter de dicho orden aparece en el vector.
La sintaxis es la siguiente:

array=count_chars(string);

nl2br

Devuelve un string en donde todos los saltos de línea se han reemplazado por el tag
 de html.

La sintaxis es la siguiente:

string=nl2br(string);

strip_tags

Devuelve un string eliminando del string original todos los tags html, si se pasa el segundo parámetro opcional es posible especificar que tags no deben eliminarse (solo hace falta pasar los tags de apertura)

La sintaxis es la siguiente:

string=strip_tags(string,string_tags_validos);

```
<?
$st2=strip_tags($st1,"<br> <table>");
//Elimina todos los tags html de $st1
//excepto <br> , <table> y </table>
?>
```

metaphone

Devuelve una representación metafónica (similar a soundex) del string de acuerdo a las reglas de pronunciación del idioma ingles.

La sintaxis es la siguiente:

string = metaphone(string);

strtok

Dado un separador obtiene el primer "token" de un string, sucesivas llamadas a strtok pasando solamente el separador devuelven los tokens en forma sucesiva o bien falso cuando ya no hay mas tokens.

La sintaxis es la siguiente:

string = strtok(separador,string);

```
<?
$tok=strtok($st,"/");
while($tok) {
        //Hacer algo
        $tok=strtok("/");
}
?>
```

parse_string

Dado un string de la forma "nombre=valor&nombre2=valor2&nombre3=valor3", asigna las variables correspondientes con los valores indicados, ejemplo:

La sintaxis es la siguiente:

parse_string(string);

```
<?
parse_string("v1=hola&v2=mundo");
//asigna $v1="hola" y $v2="mundo"
?>
```

Codificación y decodificación ASCII.

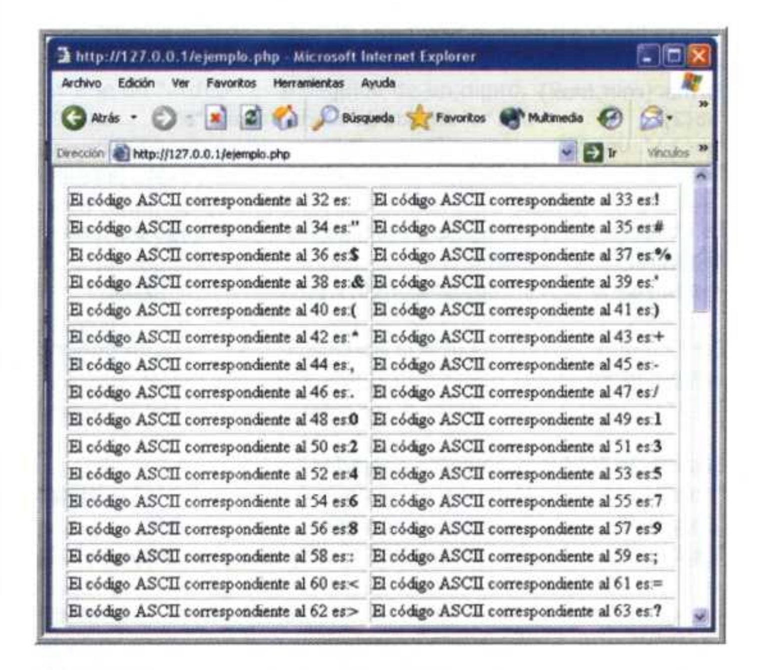
chr

Devuelve el caracter dado su número ascii.
La sintaxis es la siguiente:

char = chr(int);

```
<table border=1>
<? for ($caracter=32;$caracter<128;$caracter++) { ?>
<tr>
<td><? print "El código ASCII correspondiente al $caracter es:<b>".chr($caracter); ?></b></td>
<? $caracter=$caracter+1; ?>
<td><? print "El código ASCII correspondiente al $caracter es:<b>".chr($caracter); ?></b></td>
</tr>
<? } ?>
</table>
```

Con este sencillo código se obtiene la tabla ASCII, el resultado de ejecutar el código es el siguiente:



Ord
Dado un caracter devuelve su código Ascii. Justo lo contrario que chr()
La sintaxis es la siguiente:

int = ord(char);

Substrings:
substr
Devuelve el substring correspondiente al string pasado desde la posición indicada por offset y de la longitud indicada como tercer parámetro, si no se pasa el tercer parámetro se toman todos los caracteres hasta el final del string.

La sintaxis es la siguiente:

string = substr(string, offset, longitud);

```
<?
$cadena="Esto es un ejemplo de la función substr";
print substr($cadena,0,20); // da como resultado "Esto es un ejemplo d"
?>
```

substr_replace
Idem anterior pero el substring seleccionado es reemplazado por string_reemplazo, si string_reemplazo es "" entonces sirve para eliminar una porción de un string.

La sintaxis es la siguiente:

string=substr_replace(string,string_reemplazo,offset, longitud);

Búsquedas y Reemplazos.
str_replace
Reemplaza todas las ocurrencias de string1 en string3 por string2. Esta función no admite expresiones regulares como parámetros.

La sintaxis es la siguiente:

str_replace(string1, string2, string3);

strtr
Reemplaza en string1 los caracteres en string_from por su equivalente en string_to (se supone que string_from y string_to son de la misma longitud, si no lo son los caracteres que sobran en el string mas largo se ignoran)

La sintaxis es la siguiente:

string=strtr(string1, string_from, string_to);

```
<?
$st="hola mundo"
strtr($st,"aeiou","12345");
//$st="h4la m5nd4"
?>
```

split
Idem a explode pero el separador puede ser ahora una expresión regular.
La sintaxis es la siguiente:

array = split(pattern, string);

ereg
Devuelve true si la cadena proporcionada ha sido encontrada dentro de otra cadena. En caso contrario devolverá false. El tercer parámetro es opcional.

La sintaxis es la siguiente:

boolean = ereg(stringPatron, stringFuente,arrayRegs);

```
<?
$email=webmaster@hackxcrack.com;
if (ereg("@",$email)) {
  print("El email $email es correcto<br>");
} else {
  print("El email $email no es correcto<br>");
}
?>
```

Este ejemplo comprueba la existencia del símbolo @ en la cadena email. En una dirección de correo electrónico el símbolo @ es obligatorio, para comprobarlo utilizamos la función *ereg()*, si devuelve *true* el símbolo @ ha sido encontrado, en caso contrario devuelve false.

El tercer parámetro es opcional, consiste en un array donde se pueden almacenar las diferentes partes en las que se puede separar una cadena utilizando esta función. Una de las particularidades que tiene esta función es que también es posible utilizarla para fragmentar una cadena de texto que este bien definida. Por ejemplo en el caso de una dirección de email hay tres partes bien diferenciadas (nombre de usuario, servidor y dominio).

Utilizando el tercer parámetro de tipo *array* es posible guardar en posiciones diferentes cada una de estas partes, siendo la primera de ellas la dirección completa, la segunda el nombre del usuario, la tercera el nombre del servidor y la cuarta y última el dominio al que pertenece.

```
<?
 $email="webmaster@hackxcrack.com";
 if (ereg("^(.+)@(.+)\\.(.+)$", $email, $par)) {
   print("El email es correcto<br>");
   print("Email: $par[0] <br>");
   print("Usuario: $par[1] <br>");
   print("Servidor: $par[2] <br>");
   print("Dominio: $par[3] <br>");
 } else {
   print("El email $email no es correcto<br>");
 }
?>
```

eregi
Idem anterior pero case-insensitive.
La sintaxis es la siguiente:

boolean=eregi(stringPatron, stringFte, arrayRegs);

ereg_replace
Reemplaza todas las ocurrencias de una expresión regular en string por el contenido de string_to.
La sintaxis es la siguiente:

ereg_replace(pattern_from, string_to, string);

```
<?
 $cadena = "Hola mundo";
 $cadenaAReemplazar = "Hola";
 $cadenaReemplazada = "Adios";
 print(ereg($cadenaAReemplazar, $cadenaReemplazada, $cadena));
?>
```

eregi_replace
Idem anterior pero no considera mayúsculas y minúsculas para la búsqueda de la expresión regular en el string.
La sintaxis es la siguiente:

eregi_replace(pattern_from, string_to,string);

Sintaxis básica de una expresión regular:
Los símbolos especiales "^" y "$" se usan para matchear el principio y el final de un string respectivamente.

Por ejemplo:

"^el" — Matchea strings que empiezan con "el"
"colorin colorado$" — Matchea strings que terminan en "colorin colorado"
"^abc$" — String que empieza y termina en abc, es decir solo "abc" matchea
"abc" — Un string que contiene "abc" por ejemplo "abc" ,"gfabc", "algoabcfgeh", etc...

Los simbolos "*" , "+" y "?" denotan la cantidad de veces que un caracter o una secuencia de caracteres puede ocurrir. Y denotan 0 o más, una o más y cero o una ocurrencias respectivamente.

Por ejemplo:

"ab*" — Matchea strings que contienen una "a" seguida de cero o mas "b"

Ej: "a", "ab", "cabbbb", etc.

"ab+" — Matchea strings que contienen una "a" seguida de una o mas "b"
"ab?" — Matchea strings que contienen una "a" seguida o no de una "b" pero no mas de 1.
"a?b+$" — Matchea "a" seguida de una o mas "b" terminando el string.

Para indicar rangos de ocurrencias distintas pueden especificarse la cantidad máxima y mínima de ocurrencias usando llaves de la forma {min,max}

"ab{2}" — Una "a" seguida de exactamente 2 "b"
"ab{2,}" — Una "a" seguida de 2 o mas "b"
"ab{3,5}" — Una "a" seguida de 3 a 5 "b" ("abbb", "abbbb", "abbbbb")

Es obligatorio especificar el primer número del rango pero no el segundo. De esta forma
+ equivale a {1,}. * equivale a {0,} y ? equivale a {0,1}

Para cuantificar una secuencia de caracteres basta con ponerla entre paréntesis.
"a(bc)*" Matchea una "a" seguida de cero o mas ocurrencias de "bc" ej: "abcbcbc"

El símbolo "|" funciona como operador "or"

"hola|Hola" — Matchea strings que contienen "hola" u "Hola"
"(b|cd)ef" — Strings que contienen "bef" o "cdef"
"(a|b)*c" — Secuencias de "a" o "b" y que termina en "c"

El carácter "." matchea a cualquier otro caracter.

"a.[0-9]" — Matchea "a" seguido de cualquier caracter y un dígito.
"^.{3}$" — Cualquier string de exactamente 3 caracteres.

Los corchetes se usan para indicar que caracteres son validos en una posición única del string.

"[ab]" — Matchea strings que contienen "a" o "b"
"[a-d]" — Matchea strings que contienen "a", "b" , "c" o "d"
"^[a-zA-Z]" — Strings que comienzan con una letra.
"[0-9]%" — Un dígito seguido de un signo %

También puede usarse una lista de caracteres que no se desean agregando el símbolo "^" dentro de los corchetes, no confundir con "^" afuera de los corchetes que matchea el principio de línea.

"[^abg]" — Strings que NO contienen "a" , "b" o "g"
"[^0-9]" — *Strings que no contienen dígitos*

Los caracteres "^.[$()|*+?{\" deben escaparse si forman parte de lo que se quiere buscar con una barra invertida adelante. Esto no es válido dentro de los corchetes donde todos los caracteres no tienen significado especial.

Ejemplos:
Validar una suma monetaria en formato: "10000.00", "10,000.00" .,"10000" o "10,000" es decir con o sin centavos y con o sin una coma separando tres dígitos.
^[1-9][0-9]*$
Esto valida cualquier número que no empieza con cero, lo malo es que "0" no pasa el test.

Entonces:
^(0|[1-9][0-9]*)$
Un cero o cualquier número que no empieza con cero. Aceptemos también un posible signo menos delante.
^(0|-?[1-9][0-9]*)$
O sea cero o cualquier número con un posible signo "-" delante. En realidad podemos admitir que un número empiece con cero para una cantidad monetaria y supongamos que el signo "-" no

tiene sentido, agreguemos la posibilidad de decimales: ^[0-9]+(\.[0-9]+)?$

Si viene un "." debe esta seguido de un dígito, 10 es válido pero "10." no

Especifiquemos uno o dos dígitos decimales:
^[0-9]+(\.[0-9]{1,2})?$

Ahora tenemos que agregar las comas para separar de a miles.
^[0-9]{1,3}(,[0-9]{3})*(\.[0-9]{1,2})?$

O sea un conjunto de 1 a 3 dígitos seguido de uno más conjuntos de "," seguido de tres dígitos. Ahora hagamos que la coma sea opcional.
^([0-9]+|[0-9]{1,3}(,[0-9]{3})*)(\.[0-9]{1,2})?$

Y de esta forma podemos validar números con los 4 formatos validos en una sola expresión.

Práctica - Crear una versión de Google al estilo de Hackxcrack

La idea es crear un google pero con el diseño de Hackxcrack, para ello crearemos un pequeño script que lea el resultado de una búsqueda de google, modifique el diseño y luego muestre el resultado en el navegador.

Para leer el resultado de una búsqueda en google vamos a utilizar la lectura de un fichero remoto, ya explicado en el número anterior, ¿vas viendo la potencia que tiene este sistema de acceso remoto?, da mucho juego, ¿verdad?.

Una vez tengamos el fichero almacenado en una variable tipo string, sustituiremos la imagen de google por el de hackxcrack, se puede hacer muchas más cosas, pero esto es un ejemplo, hace un tiempo encontré una versión de Google en Pikachu utilizando el mismo sistema. Era muy curioso ver los resultados del Google en un idioma tan tonto, je je.

El código para leer el fichero de respuesta es:

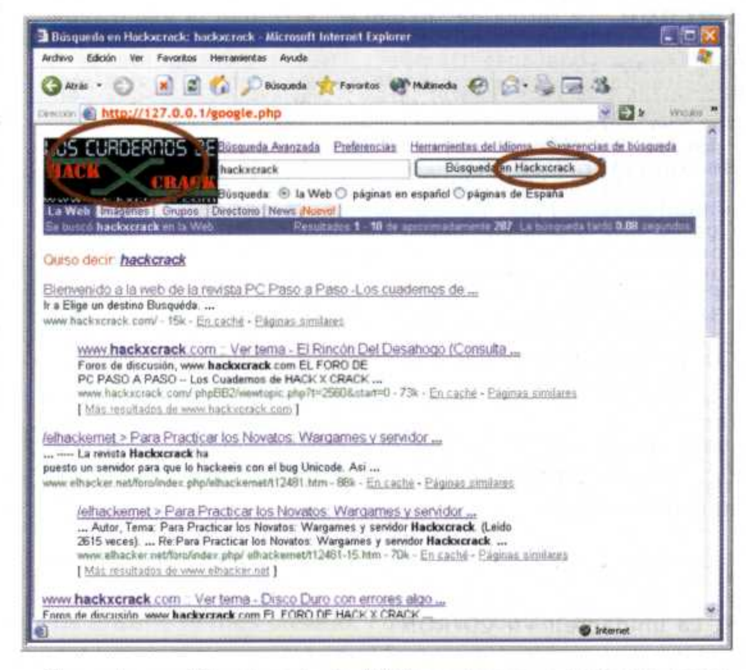
```
<?
$palabra="hackxcrack";
$busqueda="http://www.google.es/search?hl=es&ie=UTF-8&oe=UTF-8&q=".$palabra."&meta=";
if ($fp=fopen($busqueda,"r")) {
   while (!feof($fp)) {
      $pagina=$pagina.fgetc($fp);
   }
   print $pagina
}
?>
```

En la variable $pagina se encuentra almacenado el código HTML que ha generado Google en la búsqueda, al poner print $pagina lo que hacemos es que muestra la página en el navegador. Con este código mostramos en el navegador el resultado de la búsqueda, fíjate que el logo de Google no aparece, pues ese es el logo que tenemos que cambiar por el de hackxcrack para ello tenemos que hacer un cambio antes de la sentencia print $pagina.

Para ello tenemos que buscar en la variable $pagina el string web_logo_left.gif y cambiarlo por el string
http://www.hackxcrack.com/phpBB2/templates/SpacePilot3K/images/logo.gif

Para ello vamos a utilizar la función ereg_replace() antes de print $pagina, el código quedaría de la siguiente manera:

```
<?
$palabra="hackxcrack";
$busqueda="http://www.google.es/search?hl=es&ie=UTF-8&oe=UTF-8&q=".$palabra."&meta=";
if ($fp=fopen($busqueda,"r")) {
   while (!feof($fp)) {
      $pagina=$pagina.fgetc($fp);
   }
$pagina=ereg_replace("Google","Hackxcrack",$pagina);
$pagina=ereg_replace("images/web_logo_left.gif","http://www.hackxcrack.com/phpBB2/templates/SpacePilot3K/images/logo.gif",$pagina);
print $pagina;
}
?>
```



Con este sencillo ejemplo de 10 líneas hemos creado una nueva versión de Google, en donde hemos cambiado el logo y reemplazado el texto Google por el de hackxcrack. Aún se puede mejorar mucho, ya que los enlaces internos de la nueva versión de Google no funcionan, ¿te atreves a modificar toda la versión para sea 100% operativa?.

Lo importante de la práctica es que te des cuenta que con un poco de ingenio puedes lograr grandes cosas con PHP, como por ejemplo, incorporar un Google en tu web con tu propio diseño. Del mismo modo puedes capturar la información de un diario electrónico y colocarla en tu web con tu propio diseño, es necesario destacar que esto no está permitido sin la autorización previa del autor.

Bueno, por ahora ya está bien, practica y practica con las funciones de cadena y sobretodo con la lectura de ficheros remotos.

SERVIDOR DE HXC
MODO DE EMPLEO

- *Hack x Crack ha habilitado tres* servidores para que puedas realizar las prácticas de hacking.

- Las IPs de los servidores de hacking las encontrarás en EL FORO de la revista (www.hackxcrack.com). Una vez en el foro entra en la zona COMUNICADOS DE HACK X CRACK (arriba del todo) y verás varios comunicados relacionados con los servidores. No ponemos las IP aquí porque es bueno acostumbrarte a entrar en el foro y leer los comunicados. Si hay alguna incidencia o cambio de IP o lo que sea, se comunicará en EL FORO.

- Actualmente tienen el BUG del Code / Decode. La forma de "explotar" este bug la explicamos extensamente en los números 2 y 3. Lo dejaremos así por un tiempo (bastante tiempo ;) Nuestra intención es ir habilitando servidores a medida que os enseñemos distintos tipos de Hack.

- En los Servidores corre el Windows 2000 con el IIS de Servidor Web. No hemos parcheado ningún bug, ni tan siquiera el RPC y por supuesto tampoco hemos instalado ningún Service Pack. Para quien piense que eso es un error (lógico si tenemos en cuenta que el RPC provoca una caída completa del sistema), solo decirte que AZIMUT ha configurado un firewall desde cero que evita el bug del RPC, (bloqueo de los puertos 135 (tcp/udp), 137 (udp), 138 (udp), 445 (tcp), 593 (tcp)). La intención de todo esto es, precisamente, que puedas practicar tanto con el CODE/DECODE como con cualquier otro "bug" que conozcas (y hay cientos!!!). Poco a poco iremos cambiando la configuración en función de la experiencia, la idea es tener los Servidores lo menos parcheados posibles pero mantenerlos operativos las 24 horas del día. Por todo ello y debido a posibles cambios de configuración, no olvides visitar el foro (Zona Comunicados) antes de "penetrar" en nuestros servidores.

- Cada Servidor tiene dos unidades (discos duros duros):
* La unidad c: --> Con 40GB y Raíz del Sistema
* *La unidad d: --> Con 40GB*
* La unidad e: --> CD-ROM

Nota: Raíz del Servidor, significa que el Windows Advanced Server está instalado en esa unidad (la unidad c:) y concretamente en el directorio por defecto \winnt\ Por lo tanto, la raíz del sistema está en c:\winnt\

- El IIS, Internet Information Server, es el Servidor de páginas Web y tiene su raíz en c:\inetpub (el directorio por defecto)

Nota: Para quien nunca ha tenido instalado el IIS, le será extraño tanto el nombre de esta carpeta (c:\inetpub) como su contenido. Pero bueno, un día de estos os enseñaremos *a instalar vuestro propio Servidor Web (IIS) y detallaremos* su funcionamiento.

De momento, lo único que hay que saber es que cuando TÚ pongas nuestra IP (la IP de uno de nuestros servidores) en tu navegador (el Internet explorer por ejemplo), lo que estás haciendo realmente es ir al directorio c:\Inetpub\wwwroot\ y leer un archivo llamado default.htm.

Nota: Como curiosidad, te diremos que APACHE es otro Servidor de páginas Web (seguro que has oído hablar de él). Si tuviésemos instalado el apache, cuando pusieses nuestra IP en TU navegador, accederías a un directorio raíz del Apache (donde se hubiese instalado) e intentarías leer una página llamada index.html ... pero... ¿qué te estoy *contando?... si has seguido nuestra revista ya* dominas de sobras el APACHE ;)

Explicamos esto porque la mayoría, seguro que piensa en un Servidor Web como en algo extraño que no saben ni donde está ni como se accede. Bueno, pues ya sabes dónde se encuentran la mayoría de IIS (en \Inetpub\) y cuál es la página por defecto (\Inetpub\wwwroot\default.htm). Y ahora, piensa un poco... ... ¿Cuál es uno de los objetivos de un hacker que quiere decirle al mundo que ha hackeado una Web? Pues está claro, el objetivo es cambiar (o sustituir) el archivo default.html por uno propio donde diga "hola, soy DIOS y he hackeado esta Web" (eso si es un lamer ;)

A partir de ese momento, cualquiera que acceda a ese servidor, verá el default.htm modificado para vergüenza del "site" hackeado. Esto es muy genérico pero os dará una idea de cómo funciona esto de hackear Webs ;)

- Cuando accedas a nuestro servidor mediante el CODE / DECODE BUG, crea un directorio con tu nombre (el que mas te guste, no nos des tu DNI) en la unidad d: a ser posible y a partir de ahora utiliza ese directorio para hacer tus prácticas. Ya sabes, subirnos programitas y practicar con ellos :) ... ... ¿cómo? ¿que no sabes crear directorios mediante el CODE/DECODE BUG... repasa los números 2 y tres de Hack x Crack ;p

Puedes crearte tu directorio donde quieras, no es necesario que sea en d:\mellamojuan. Tienes total libertad!!! Una idea es crearlo, por ejemplo, en d:\xxx\system32\default\10019901\mellamojuan (ya irás aprendiendo que cuanto mas oculto mejor :)

Es posiblemente la primera vez que tienes la oportunidad de investigar en un servidor como este sin cometer un delito (nosotros te dejamos y por lo tanto nadie te perseguirá). Aprovecha la oportunidad!!! e investiga mientras dure esta iniciativa (esperemos que muchos años).

- En este momento tenemos mas de 600 carpetas de peña que, como tu, está practicando. Así que haznos caso y crea tu propia carpeta donde trabajar.



MUY IMPORTANTE...

MUY IMPORTANTE!!!!! Por favor, no borres archivos del Servidor si no sabes exactamente lo que estás haciendo ni borres las carpetas de los demás usuarios. Si haces eso, lo único que consigues es que tengamos que reparar el sistema servidor y, mientras tanto, ni tu ni nadie puede disfrutar de él :(Es una tontería intentar "romper" el Servidor, lo hemos puesto para que disfrute todo el mundo sin correr riesgos, para que todo el mundo pueda crearse su carpeta y practicar nuestros ejercicios. En el Servidor no hay ni Warez, ni Programas, ni claves, ni nada de nada que "robar", es un servidor limpio para TI, por lo tanto cuídalo un poquito y montaremos muchos más :)

SERIE XBOX LIFE(III)

CAMBIANDO EL DISCO DURO Y PARCHEANDO JUEGOS.

- Cambiamos el Disco Duro de la XBOX
- Parcheando los Juegos que te bajas de Internet ;)

Muy buenas a todos. Este mes vamos a ver cómo cambiar el Disco Duro de la consola (HD) y vamos a parchear juegos.

Primero quisiera que quedase claro que si cambiáis el HD de la consola, no podréis jugar online con el servicio de Xbox live, si queréis usar este servicio debéis usar el HD original que viene con la consola. Si usáis un HD que no es de la consola o intentáis acceder al servicio online de Xbox con una copia de seguridad o con el Mod chip activado seréis expulsados del servicio, hay rumores de que este "baneo" es sólo por una temporada, pero no lo he comprobado.

Ahora os dejo una lista con lo que necesitamos para seguir este artículo:

Para cambiar el HD de la consola:

Xbox con Modchip y Evolution x instalado.

Evolution x.

Un destornillador Torx 20.

Un destornillador Torx 10.

Cd-Rw.

Disco duro que queramos poner a la consola.

Para HD mayores de 120GB habrá que tener una bios que lo acepte, ya hablaremos de eso.

Pc preparado.

Para parchear juegos:

El juego a parchear.

ZXB Tools 1.4.

Los que os hayáis leído los artículos anteriores, sabréis que ahora debería explicar lo que es cada cosa, pues este mes me lo salto, casi todo lo que necesitamos lo hemos visto en artículos anteriores, si no tenéis los 2 números anteriores los podéis pedir en la web.

Los programas y las bios para los HD mayores de 120Gb los encontraréis en la web.



Cuando digo...

Cuando digo Pc preparado, lo que quiero decir, es que debéis tener un Pc, con tarjeta de red, grabadora de Cd´s o Dvd´s, con el Nero 6.0.0.19 o superior instalado, un cliente FTP y uno o dos cables JR45 según hagáis la conexión. Todo esto se explicó en números anteriores.

Cambiando el disco duro:

Lo primero que debemos hacer, es una copia de seguridad de nuestro HD de la consola. Para eso usaremos el Evo-x configurado como os enseñé el mes pasado. Podéis encontrar el contenido del evox.ini en el foro y en la Web..

Encendemos la consola y nos dirigimos al apartado de "utilidades de sistema" hay una opción llamada "copia de seguridad", le damos y nos creará una carpeta con el nombre "backup" en la partición C del HD de la consola. Ahora nos conectamos vía FTP y copiamos el contenido de la partición C a nuestro Pc.

También deberíais copiar el contenido de las partición E y F si es que tenéis algo importante en estas particiones como, por ejemplo, las partidas guardadas, que se encuentran en partición E y son las carpetas "TDATA" y "UDATA", así que las pasamos al PC.

Ahora debemos crear una carpeta en nuestro Pc que se llamará, por ejemplo, "nuevohd". Aquí copiaremos y configuraremos un Evolution x, puede ser el que tenéis instalado *en la consola (en este caso copiáis la carpeta* SKINS y los archivos EVOXDASH.xbe y EVOX.ini) u otro que encontraréis nuestra web. Editamos el evox.ini con el bloc de notas tal como os enseñé el mes pasado. Todo lo que haya bajo [Menu] lo borramos y ponemos esto:

```
Section "Root"
{
Item "Instalar disco duro",@210
Item "Configuracion",@9
Item "Reiniciar",@5
}

[Action_10]


Info "Formatear Disco"
Warning "Estas seguro?"
Progress "Formateando.... =)"

ConfigSector "\backup\disk.bin"
Format c:
Format e:
Format f:
Format x:
Format y:
Format z:
```

Muy importante, hay que dejar una línea en blanco al final del archivo (justo debajo de "Format z:"), si no es así no funcionará.

Configurar los valores de la red según la vuestra, como os expliqué el mes pasado.

Guardamos el evox.ini, y cerramos el bloc de notas.

Copiamos la carpeta backup, antes creada por el Evolution x, a la carpeta nuevohd, que es donde tenemos el Evox recién configurado. Grabamos todo esto con el nero, tal y como deberíais saber.

Ahora vamos a abrir la consola. Para esto colocamos la consola boca abajo, quitamos las cuatro gomas de las esquinas, y veremos los tornillos. Usamos el destornillador *Torx 20 para quitarlos, hay dos mas,* uno debajo de la pegatina blanca, donde pone la fecha de fabricación y otro debajo de la pegatina holográfica. Ahora le damos la vuelta y quitamos la parte superior: veremos a mano izquierda el lector DVD y a mano derecha el HD, en el centro hay un tornillo que deberemos quitar con el destornillador Torx 10 y hay otro más, que quitaremos con el mismo destornillador, que está cerca de la esquina inferior derecha del lector DVD.

Ahora desenchufamos el cable de la fuente de alimentación y el cable IDE del HD y levantamos todo el módulo del HD. El DVD se quedará en su sitio. En los laterales del módulo del HD hay 2 tornillos de estrella, los quitamos y retiramos el HD del módulo de plástico.

Ponemos el nuevo HD en modo master, lo atornillamos al módulo de plástico y volvemos a montarlo todo.

Encendemos la consola con el Mod chip activado. El led parpadeará verde rojo, es normal, metemos el CD-RW que acabamos de quemar, apagamos y volvemos a encender (con el Mod-chip activado). Arrancará el Evox, le daremos a la opción "instalar disco duro", confirmamos, y se habrán formateado todas las particiones y estarán listas para usar.

Reiniciamos la consola con el chip activado y con el Cd aún dentro. Vamos al Pc y nos conectamos vía FTP a la consola. Ahora hay que volver a poner las cosas en su sitio, es decir, lo que había en la partición C del antiguo HD, *lo tendremos que copiar al nuevo, y así* con todo.

Y con esto ya está hecho. Sacamos el Cd, reiniciamos la consola y veremos que arranca normalmente.

Ahora paso a explicar los HD mayores de 120 Gb:

Para poder usar discos duros mayores de 120Gb habrá que cambiar la bios del chip para que acepte más tamaño. Hay dos bios que cumplen esta función y las podéis encontrar en la web. Lo que no puedo deciros es cómo cambiar la bios de todos los chips del mercado, así que deberéis buscar en Google información sobre el tema. Después de tener las bios actualizadas, seguís los pasos que acabo de explicar.

Las bios son las rutinas que lee la consola al arrancar. Dependiendo del chip que tengáis, se podrá o no cambiar la bios, y según sea el chip, se cambia la bios de una u otra forma. A veces es tan fácil como copiar la bios a la carpeta "bios" en la partición C del HD de la consola. Con el Evolution x, en el apartado de "utilidades de sistema", está la función de "flashear bios". Elegimos esta opción y seleccionamos la bios que queramos, confirmamos y se cambiará la bios. Reiniciamos y todo debería ir bien.

Pero no todos los chip aceptan esta función, por ejemplo, el Aladin advance no lo acepta. Hay que configurar el Xbox de determinada forma. Como es algo complicado de explicar, he colocado en la web de hxc el archivo llamado "Aladin Cd Flasher.rar".

Sólo hay que descargarlo, descomprimirlo y grabarlo con el Nero en un CdRW.

Elegimos la opción "Flashsear bios" y seleccionamos la bios que más nos convenga:

Bios 4981.67 : tienes unidades F y G (en el Evolution X de la consola hay que cambiar el valor de la opción "useGdrive" a YES -por defecto está en NO).

Bios 4981.06 : tienes todos los gigas en la unidad F.

Hay otros chip que vienen con programador y en estos es más fácil y fiable cambiar la bios, ya que si lo flasheamos mal, lo volvemos a flashear con otra bios y debería funcionar.

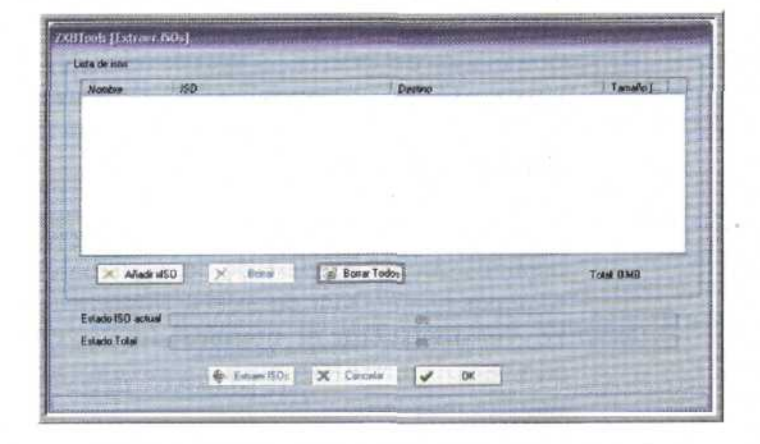
Debéis conocer bien vuestro chip y estar convencidos de lo que vais a hacer. Si flasheais mal el chip, puede que no vuelva a funcionar.

Parcheando juegos:

Primero nos bajamos ZXBTOOLS de la web y lo descomprimimos en el Pc. Este programa carece de instalación y lo podemos arrancar sin más.

Ahora nos ponemos en situación: nos hemos bajado un juego, pero está en una iso que ninguno de tus programas abre y crees que esta corrupto porque no está en formato Xiso "only for xbox". Así que arrancamos el ZXBTOOLS y nos sale esto:

Pinchamos en ISO> Extraer y nos sale esta imagen:



Ahora pinchamos en añadir xISO, seleccionamos la iso que nos hemos bajado de Internet, indicamos dónde queremos que se extraiga la iso y pulsamos en aceptar. Luego pinchamos en "Extraer Isos", esperamos un rato y ya estará extraído.

Muy fácil, no?

Pues ahora lo parcheamos. Pinchamos en Utilidades> Parchear XBE



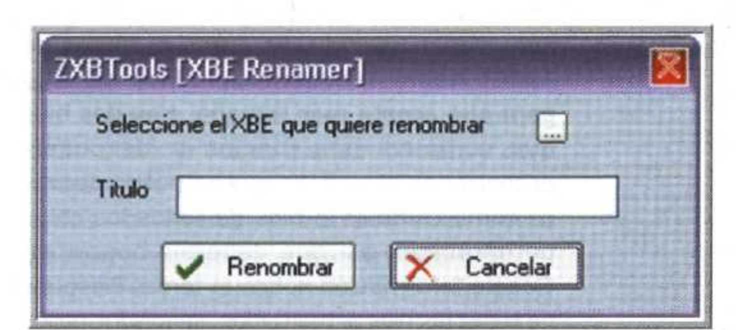
Nos aseguramos de que las dos casillas estén activadas y seleccionamos el default.xbe que deseamos parchear. Pulsamos en parchear y en el cuadro de texto podréis ver si se ha parcheado o no.

Con esto ya estaría listo el juego para grabar con el nero. Esto de parchear no lo necesitan todos los juegos, pero es recomendable que lo hagáis cada vez que grabéis un juego.

Otra función del ZXBTOOLS es la de renombrar los xbe, esto no significa renombrar el archivo, sino el nombre del archivo que se lee en la consola.

Esto lo entenderemos mejor con un ejemplo: digamos que en el apartado de emuladores tenemos el emulador de SUPER NINTENDO. Lo veríamos con el nombre de SUPER NINTENDO ENTERTAIMENT SYSTEM, un nombre muy largo y feo :).

Nos vamos a ZXB TOOLS, pinchamos en Utilidades> renombrar XBE



Seleccionamos el XBE, Ponemos el nombre que queramos, por ejemplo SNES, en el caso de este emulador, y pinchamos en renombrar. Saldrá un mensaje de confirmación, diciendo que se ha renombrado adecuadamente el archivo y ya tendríamos el nombre cambiado.

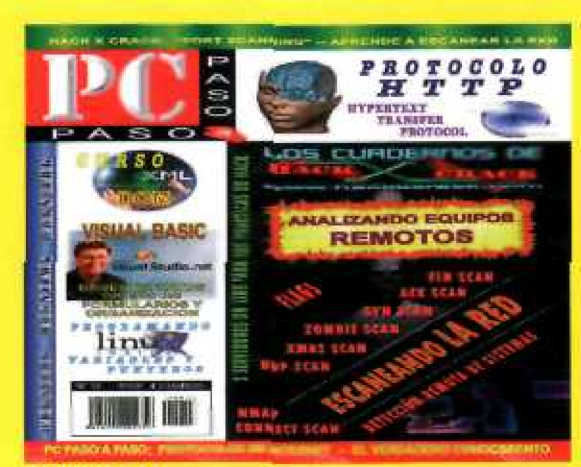
Con esto hemos acabamos este mes. Sólo me queda decir que el mes que viene instalaremos el programa para ver DIVX en nuestra consola, descubriremos los distintos lectores de DVD que tiene la XBOX y algunas cosas más.

Salu2

Pd: Tengo que dar las gracias a BIO por ayudarme con el tema de flashear la bios del Aladin y pasarme el Aladin Cd Flasher.



NÚMERO 12:
- Curso de linux: programacion C.
- Visual Basic: IIS bug exploit. Nuestro primer Scanner.
- APACHE: Configuralo de forma segura.
- Serie Raw: FTP(II)
- VALIDACION XML: DTD (II)



NÚMERO 13:
- Curso de linux: programacion C(II).
- Visual Basic:Nuestro primer proyecto.
- APACHE: Configuralo de forma segura.
- Serie Raw: HTTP.
- CURSO XML: DOM.



NÚMERO 14:
- Curso de linux: programacion C(III).
- Visual Basic:Nuestro primer proyecto(II).
- Curso de PHP
- Serie Raw: DNS.
- CURSO XML: DOM(II).
- HIJACKING



NÚMERO 15:
- CURSO DE PHP (II)
- Xbox. Instalar Linux
- SERIE RAW (9): MSN
- CURSO VISUAL BASIC: UN CLIENTE, UNA NECESIDAD(III).
- PROGRAMACION BAJO LINUX: LENGUAJE C(III)



NÚMERO 16:
- CURSO DE PHP (III)
- Xbox. Evolution X
- SERIE RAW (10): USENET
- CURSO XML: DOM(III).
- PROGRAMACION BAJO LINUX: EL SISTEMA IPC.

CONSIGUE LOS NUMEROS ATRASADOS EN:

WWW.HACKXCRACK.COM